# Verbal Chess using Computer Vision with the Baxter Research Robot

Design Team: Connor Desmond, Zephaniah Connell, and Ryan Cook
Advisors: Dr. Suresh Muknahallipatna and Deborah Kretzschmar
Supervisor: Victor Bershinsky
Date: May 2, 2018

EE4830-Senior Design II Final Report
Department of Electrical and Computer Engineering
College of Engineering and Applied Science
University of Wyoming

**Abstract**
The Baxter robotic system is an extremely sophisticated piece of machinery, equipped with a myriad of sensors and features. As of yet, very little research has been accomplished utilizing Baxter by students or faculty in the Department of Electrical and Computer Engineering at the University of Wyoming. This project is a base that will enable future employment of Baxter for more intricate and advanced research topics. This project was derived to showcase a large portion of Baxter's functionality in an easily digestible and potentially expandable format. It will display a convenient form of user interaction (voice commands), utilization of computer vision (detecting chess board and pieces), and safe and precise physical interactions with or near humans (moving chess pieces).
The goal of this project is to enable the Baxter robotic system to move chess pieces on a chess board based on user input in the form of voice commands. This can be broken up into four main parts: physical movement of Baxter's appendages, computer vision to locate the board and chess pieces, voice recognition and speech output for the necessary set of commands, and internal chess board state information and chess logic.

## Background Information

Baxter has two arms with seven degrees of freedom; each of which is equipped with a pair of grippers and a high definition camera on the end. The Baxter Research Robot SDK interfaces with Baxter via the Robot Operating System (ROS) and its various ROS APIs. This project utilizes the camera on the end of the left arm to capture the chess board and pieces. Prewritten Python and ROS libraries, including pocketsphinx, Baxter's Inverse Kinematics Service, pyttsx, and OpenCV are employed to enable computer vision, speech recognition, speech output, and arm movements. This project is intended to be a stepping stone for future applications and a proof of concept of the Baxter Research Robot's capabilities and use as many Baxters abilities to play a game of chess, through our learning of ROS. The only things that changed from the original design goals were a team member change and a slight change in how the computer vision worked, in that baxter did not find unique pieces just the closest piece to the proper location of the piece.

## Functional and Technical Specifications

### Baxter Research Robot

Baxter has two seven degree-of-freedom (DOF) arms that utilize Series Elastic Actuators. It has an arm span of 261 cm and a height of 185 cm. On the end of each arm is a pair of replaceable grippers. The only actions the grippers are capable of are opening and closing (as the grippers are only composed of two digits). Baxter is equipped with a camera on the end of both arms. The Baxter Research Robot SDK provides a software interface for developing custom applications to run on the Baxter platform. The SDK interfaces with Baxter via the Robot Operating System (ROS) and its various ROS APIs.

### Operation

The 'README.md' file on this team's GitHub describes the necessary operations to initialize and operate this project. The following will describe the details of the operations that were created in order to make Baxter play chess.

### Computer Vision and Movement: Initialization

The initialization of Baxter was done through the running of a camera control tool that would open and close the left hand camera in a specific order that way the camera was set up to a resolution of 960x600. This was done so the computer vision algorithms knew the proper pixel count for further usage. The next step was to then have Baxter hover above the chess board at a height of 62 cm then view the space and find the largest enclosed shape, which happened to be the chess board. Baxter would then approach the middle of the board and locate the corners. This process was all defined under a method called CannyIt. Lastly a formula was derived to align the picture coordinates with Baxters coordinates so the middle of each square on the chess board could be turned into a pose that Baxter would eventually move to.

### Speech Recognition

Speech recognition was accomplished utilizing a package developed by the Carnegie Mellon University Sphinx team called 'pocketsphinx.' A text file, 'speech_commands.txt' included in appendices, was created with a list of all commands that needed to be recognized for operation of the chess game. A handful of other files were generated from this file using an online tool, which were utilized by the main script for speech recognition, 'recognizer.py.' This script does the heavy lifting for integration with the default microphone input of the connected computer and determines whether any of that input matches a word or phrase from the dictionary text file. If there is a match, the script publishes that information to the ROS Topic '/recognizer/output/'. ROS Topics are essentially streams of

information maintained by the master node (the Baxter robot, itself) to and from which other nodes (scripts that are run connected to the master node) can post and access information through the standard form of ROS message data types.

**Speech Output**

Speech output works in a similar way to speech recognition. A server code is run, which connects to the default audio output device of the computer linked to the Baxter Research Robot. A client script then connects to the server through a three-way TCP handshake, and indicates the desired voice output with a function whose input is the desired voice output as a python string. That string is then converted by the server to an audio, speech file utilizing the pyttsx, python text to speech, library and played by the connected speaker. Speech output is utilized to indicate to the user the state of the chess game and the next expected voice input. All of the outputs utilized can be found in the 'voicecommands.py' script included in the appendices as all the definitions of the variable 'what_to_say.'

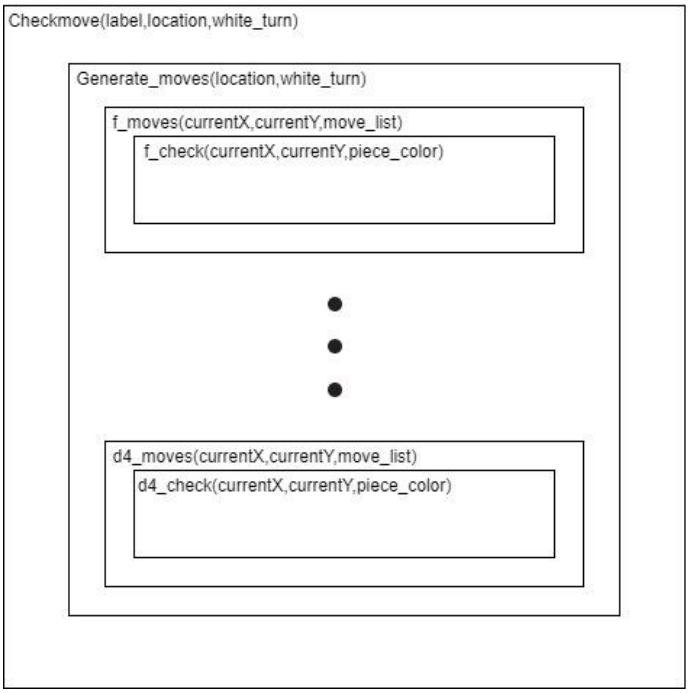
**Computer Vision and Movement: Pieces**

The next step as far as computer vision was concerned was how to properly pick up a piece to move from one square to the next. This was done through approaching the pose that was determined in initialization and hovering over that area, this in effect would be close to being directly over the desired piece. Then Baxter would take another picture and using a method called findContours, the piece would be found by identifying the color difference between the piece and its background. After the piece is found a moment calculation was performed to find the center of the piece and converted from pixel coordinates to Baxter's coordinates. Once Baxter has the new coordinates of the exact location of the piece the pose is updated and the move is performed. The move is performed by calling Baxter's IK servers that link two poses, this service figures out the best way to get from one position to the next so the exact movement between poses could vary. For this reason a middle ground coordinate was programmed in that would make sure Baxter was always moving away from the board in order to preserve its correct state. The last step to perfecting this method was determining the offsets and the meters per pixel of each picture to try and fine tune piece location as much as possible.

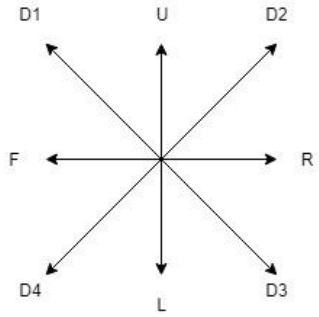
**Virtual Chess Board and Logic**

The goal of the chess logic was to maintain a virtual representation of the physical chess board. This was needed so that two main functionalities could be achieved. Primarily, there needed to be a check on whether or not a proposed move was valid. This included checking on whose turn it was, what kind of piece was being moved, and where it was being moved to. The generic algorithm used to perform this verification is shown in the figure below. Once a move was determined to be valid Baxter would perform the physical movement. Afterwords, the user would verify that the move has successfully been completed. Secondly, to ensure the parallelism of the virtual board and physical board the state of the virtual board needed to be updated after the physical piece was moved. This involved swapping important object data members from one location to another. For example, Piece A at Location 1 is moved to Location 2. The information that was originally at location 2 is overwritten by the information that was at Location 1. Then the information at Location 1 is overwritten with the information of the "Null" piece. The "Null" piece is a virtual construct which is used to represent an empty square on the board. Refer to the code appendix for the actual Python implementation of the chess logic.

Piece Movement Validation Functional Abstraction




Piece Movements

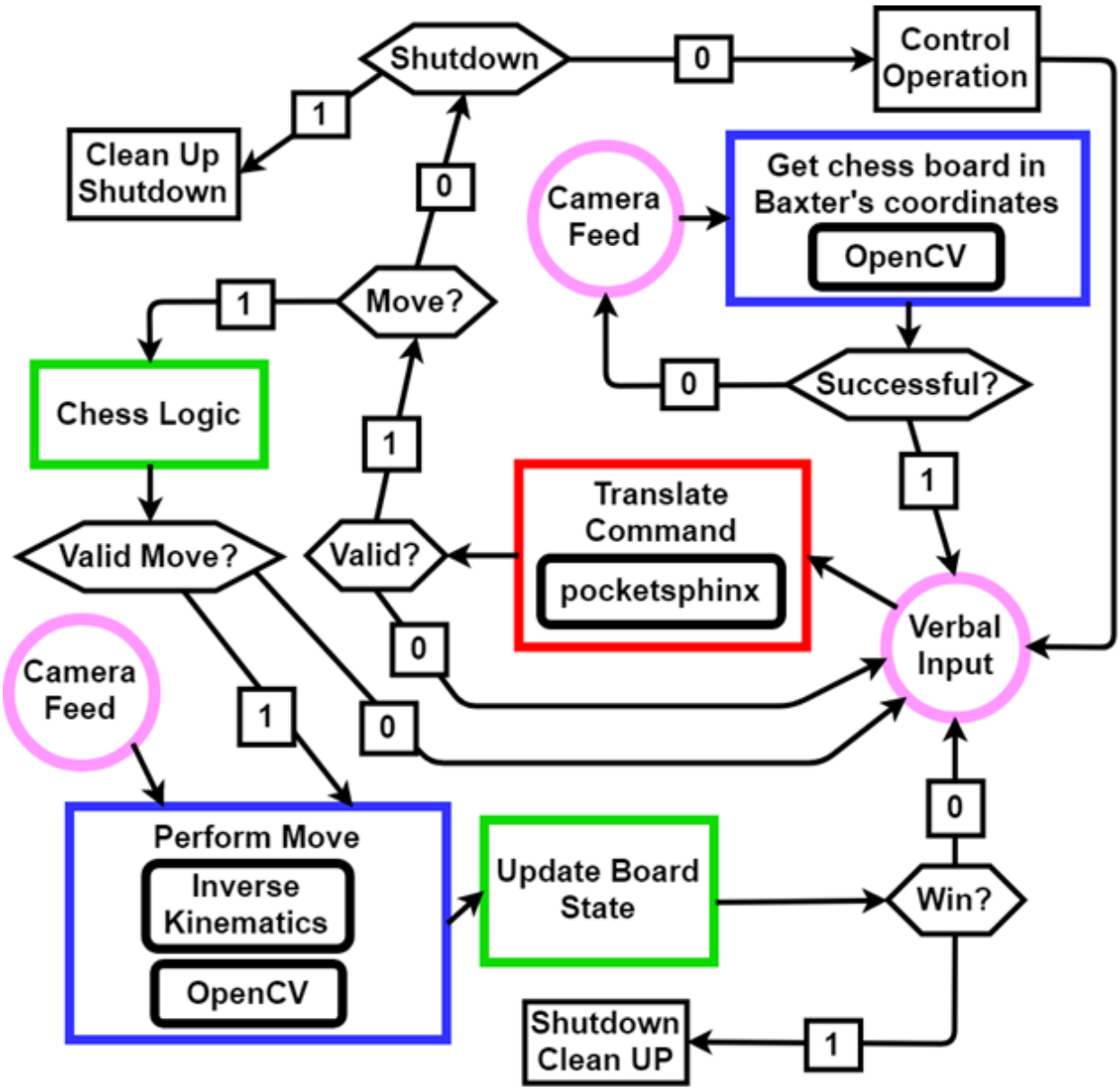



**Component Integration**

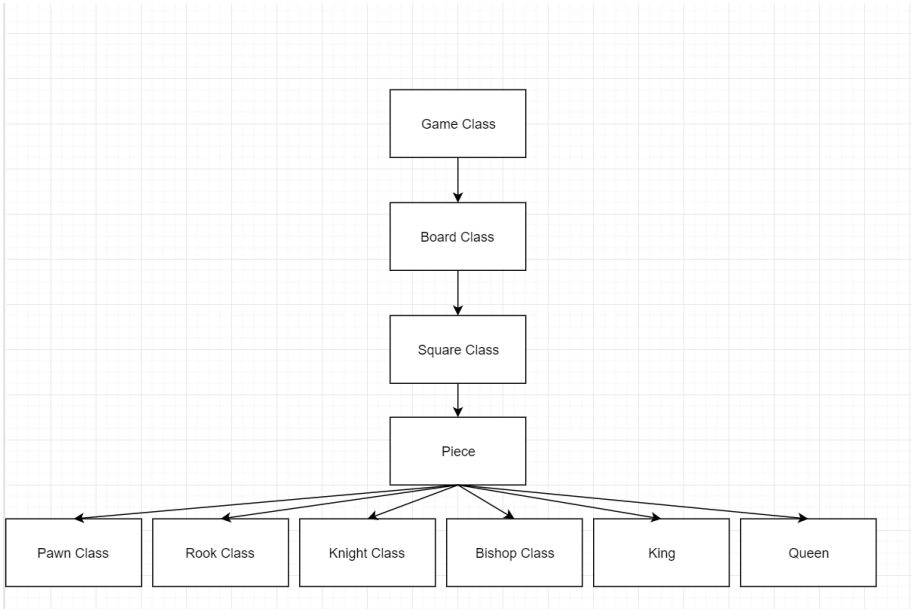
All the various components of this project were developed independently of each other and combined later to make a cohesive whole. The original design was to integrate everything utilizing the ROS Topic utility; however, each separate portion of code was constructed largely using classes that could be easily imported and utilized in other scripts. This enabled the use of one main, driver code with imports of the various developed classes, providing all the necessary functionality to integrate and execute the entire project as desired. As such, because the direction of execution is determined by voice commands, that became the base code in which to import all the other classes. The 'voicecommands.py' script immediately executes the board location initialization, then executes all other operations (piece movement, functional operations, move validation, and updating virtual board state) as determined by receipt of valid voice commands from the user.

**Software Flow Chart and Schematics**

This diagram includes the main sections of code (as the rectangles), inputs into the system (as the Circles), and decisions in the code logic (as the hexagons). The utilized libraries are also included is the rounded rectangles inside the blocks of code in which they will be utilized.



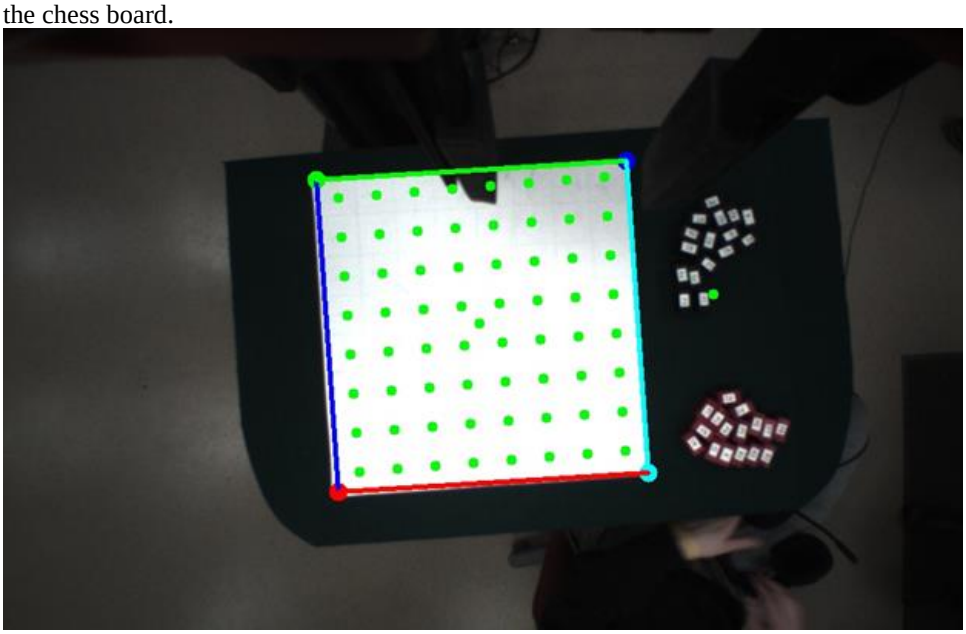

Object Oriented Design of Chess Logic:




## Results

### Computer Vision and Movement: Initialization

The board is found and the corners are properly found and placed as coordinates at an error range of .007m, which gives each center of the square an error of about 1.2cm in any direction. Below shows the output to the user on how the computer vision should look for a successful finding of the board and marking of coordinates for each square on the chess board.



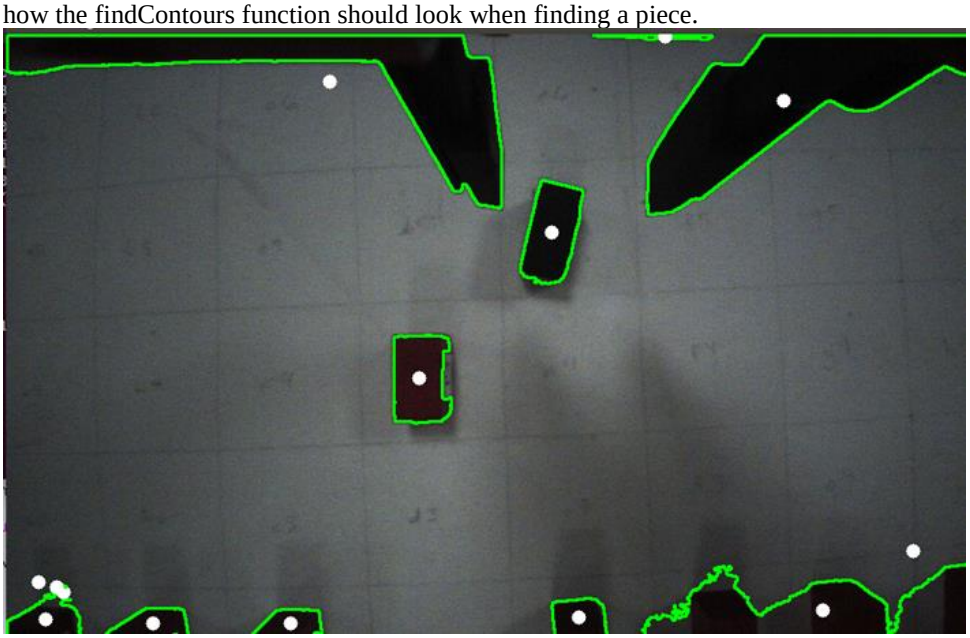
### Speech Recognition

Speech recognition can become inconsistent under the following conditions: there is background noise interfering with the quality of voice input, words included in the recognition library are overly simple or similar, or the enunciation of the user is poor due to light consonant sounds, uneven volume, and very low or high pitched voices. Our command library was changed to use the NATO phonetic alphabet (alpha, bravo, charlie, delta, echo, foxtrot, golf, and hotel) rather than the letters a-h, as well as the response of ‘success’ or ‘failure’ rather than ‘yes’ or ‘no’ to

minimize the occurrence of incorrect interpretations and false positives of commands. Outside of the aforementioned conditions, speech recognition accuracy is upwards of 95%.

**Speech Output**
As the speech output technique was relatively simple, it consistently worked properly. The only issue is that if the output volume is increased too much, it can affect the input of voice commands.

**Computer Vision and Movement: Pieces**
Piece movement took a lot of trial and error to properly pick up a piece. Through experimentation it was found that Baxter was on average about 8cm off in the Y direction and 4cm off in the X direction from where he thought he was at a height of 32cm from the board. These offsets were then sent into the conversion function that changed picture pixel coordinates to Baxter coordinates. at a height of 16cm the Y offset was 3.3cm and the x was still 4cm. The pieces were found using a findContours function that would just look for the color difference of each pixel and note the change as a coordinate that was used for a moment calculation and a canter value of the piece. Below shows how the findContours function should look when finding a piece.



Notice that multiple shapes are found, the piece that is picked should have a center closest to the center of the picture keeping in mind of the current offsets from the grippers to the center of the picture. The movement worked as it should once Baxter had the correct coordinates and pose to move to.

**Virtual Chess Board and Logic**
The goal of the virtual chess board and logic component of this project was to accurately represent the physical layout of the chess board virtually using the Python programming language. Using this created framework the important tasks such as move validation, killing a piece, when the game is over, and updating the state of the virtual board could be answered. The result to this work was a complete success. Every task that was implemented in the using the chess logic code worked without bugs or errors. Below is an example of output that my code can produce in the terminal which was used multiple times for debugging purposes.

```
In [1]: runfile('E:/Senior Design/Chess Code/Square_Driver.py', wdir='E:/Senior Design/Chess Code')
Rook Knight Bishop Queen King Bishop Knight Rook

Pawn Pawn Pawn Pawn Pawn Pawn Pawn Pawn

Null Null Null Null Null Null Null Null

Null Null Null Null Null Null Null Null

Null Null Null Null Null Null Null Null

Null Null Null Null Null Null Null Null

Pawn Pawn Pawn Pawn Pawn Pawn Pawn Pawn

Rook Knight Bishop Queen King Bishop Knight Rook

Black Black Black Black Black Black Black Black

Black Black Black Black Black Black Black Black

None None None None None None None None

None None None None None None None None

None None None None None None None None

None None None None None None None None

White White White White White White White White

White White White White White White White White
```

**Component Integration**

The entire projects works together properly as desired. All goals, after slight modification as a result of losing a team member, were successfully accomplished. User interaction through voice commands and responses was successfully implemented, computer vision paired with physical movements work properly, and the chess logic is accurate and successfully integrated into operation.

## Future Considerations

Due to time constraints there were some areas of this project where similar projects could improve on our progress. This includes using a more robust computer vision algorithm which would be more resistant to variations in ambient light as well as being able to detect more general shapes of objects in order to differentiate between chess pieces and enable the ability to initialize the project with a chess game that is already in progress. One other improvement that could be made is to utilize the MoveIt movement library, rather than Inverse Kinematics; MoveIt can include and avoid physical obstacles when deciding how to move from one location to another.


## Acknowledgements

A special thanks to the following people who contributed heavily to making this project possible: Deborah Kretzschmar, Dr. Suresh Muknahallipatna, and Victor Bershinky.


## Appendices

Unused code, reference files, and old versions of files can be found in the Github repository located at the following address: https://github.com/zephconnell/Baxter_Chess

This repository can also be utilized as a reference for the directory structure of the project as implemented. The 'Baby Steps' folder in the repository (implemented as 'baby_steps' on the linux computer connected to the Baxter Research Robot) specifically is the final version of the ROS workspace package that was utilized for this project. All code and files utilized for final operation of this project are included in the appendices below.

**References and General Functionality Code**

**README.md -**

How to run Open terminal
cd ros_ws ./baxter.sh roslaunch baby_steps speech_commands.launch
new terminal
cd ros_ws ./baxter.sh rosrun baby_steps pyttsx_server.py
new terminal
cd ros_ws ./baxter.sh rosrun baby_steps camera.py rosrun baby_steps voicecommands.py

**license.txt -**

**MULTI-SOURCE LICENSE**

**This package contains a "mish-mosh" of ideas and code snippets from several sources. This includes Rethink Robotics, "ROS by Example" by Patrick Goebel, "Programming Robots with ROS" by Morgan Quigley, Brian Gerkey and William Smart, ROS Tutorials about the turtlesim and transformation, Turtlesim tutorials by Professor Anis Koubaa, and the Robotics Laboratory at Sun Yat-sen University called the Biomimetics and Intelligent Robtics Lab (BIRL)run by Dr.Rojas. The BIRL robotics lab has a very helpful wiki which contains tutorials and shows their work. In addition, the Neuroscience and Robotics lab at Northwestern University had a demo using Baxter. Their idea to use an adaptive double exponential smoothing filter and some vector operations were used in this package. Although the information used is open source, some had different types of open source licenses....so several are listed in this text. They licenses listed below in order:**

**(1) Rethink Robotics**
**(2) "Programming Robots with ROS" by Morgan Quigley et al.**
**(3) "ROS by Example" by Patrick Goebel**
**(4) ROS Tutorials**
**(5) Use of pyttsx a cross-platform Python wrapper for text-to-speech synthesis**
**It was written by Peter Parente**
**(6) CMU Sphinx (Carnegie University Sphinx) at http://cmusphinx.sourceforge.net/**
**(7) Micheal Ferguson wrote the ROS pocketsphinx package for speech recognition.**

---

**BELOW IS THE RETHINK ROBOTICS COPYRIGHT NOTICE.**



**Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:**

**1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.**
**2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.**
**3. Neither the name of the Rethink Robotics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.**

**THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.**

**The link to the GNU General Public License is below:**

**http://www.gnu.org/licenses/gpl-3.0.en.html**

---

**BELOW IS THE LICENSE FOR THE CODE USED FROM PROGRAMMING ROBOTS WITH ROS**

**Apache License**

**Version 2.0, January 2004**

**http://www.apache.org/licenses/**

**TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION**

**1. Definitions.**

**"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.**

**"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.**

**"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.**

**"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.**

**"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.**

**"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.**

**"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).**

**"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.**

**"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."**

**"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.**

**2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.**

**3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.**

**4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:**

**(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and**

**(b) You must cause any modified files to carry prominent notices stating that You changed the files; and**

**(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and**

**(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.**

**You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.**

**5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed**

**with Licensor regarding such Contributions.**

**6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.**

**7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions**

---

**of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.**

**8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.**

**9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.**

**END OF TERMS AND CONDITIONS**

**APPENDIX: How to apply the Apache License to your work.**

**To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "{}" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.**

**Copyright {yyyy} {name of copyright owner}**

**Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at**

**http://www.apache.org/licenses/LICENSE-2.0**

**Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.**

**HERE IS THE LICENSE TO USE PYTTSX**
**pyttsx Copyright (c) 2009, 2013 Peter Parente**

**Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.**

**THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.**

---

**ROS BY EXAMPLE BY PATRICK GOEBEL USES A BSD LICENSE. IT IS BELOW**

**Copyright (c) <2016>, <Patrick Goebel>**
**All rights reserved.**

**Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:**

- **Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.**
- **Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.**
- **Neither the name of the <organization> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.**

**THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL <COPYRIGHT HOLDER> BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.**

---

**Use of pyttsx a cross-platform Python wrapper for text-to-speech synthesis**
**It was written by Peter Parente**

**pyttsx Copyright (c) 2009, 2013 Peter Parente**

**Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.**

**THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.**

---

**PocketSphinx License**

**Copyright (c) 1999-2016 Carnegie Mellon University. All rights reserved.**

**Redistribution and use in source and binary forms, with or without**

**modification, are permitted provided that the following conditions are met:**

**1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.**

**2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.**

**This work was supported in part by funding from the Defense Advanced Research Projects Agency and the National Science Foundation of the United States of America, and the CMU Sphinx Speech Consortium.**



---

**Michael Ferguson's package for pocketsphinx**

**http://wiki.ros.org/pocketsphinx. License is BSD**



**package.xml -**

```
<?xml version="1.0"?>
<package>
  <name>baby_steps</name>
  <version>0.0.0</version>
  <description>The baby_steps package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="baxter@todo.todo">baxter</maintainer>
```

```
  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>


  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/baby_steps</url> -->


  <!-- Author tags are optional, multiple are allowed, one per tag -->
  <!-- Authors do not have to be maintainers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->


  <!-- The *_depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <!-- Examples: -->
  <!-- Use build_depend for packages you need at compile time: -->
  <!--   <build_depend>message_generation</build_depend> -->
  <!-- Use buildtool_depend for build tool packages: -->
  <!--   <buildtool_depend>catkin</buildtool_depend> -->
  <!-- Use run_depend for packages you need at runtime: -->
  <!--   <run_depend>message_runtime</run_depend> -->
  <!-- Use test_depend for packages you need only for testing: -->
  <!--   <test_depend>gtest</test_depend> -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>actionlib</build_depend>
  <build_depend>baxter_core_msgs</build_depend>
  <build_depend>baxter_interface</build_depend>
  <build_depend>control_msgs</build_depend>
  <build_depend>cv_bridge</build_depend>
  <build_depend>dynamic_reconfigure</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>sensor_msgs</build_depend>
  <build_depend>trajectory_msgs</build_depend>
  <build_depend>xacro</build_depend>
  <build_depend>message_generation</build_depend>
  <run_depend>actionlib</run_depend>
  <run_depend>baxter_core_msgs</run_depend>
  <run_depend>baxter_interface</run_depend>
  <run_depend>control_msgs</run_depend>
  <run_depend>cv_bridge</run_depend>
  <run_depend>dynamic_reconfigure</run_depend>
  <run_depend>rospy</run_depend>
  <run_depend>sensor_msgs</run_depend>
  <run_depend>trajectory_msgs</run_depend>
  <run_depend>xacro</run_depend>
  <run_depend>message_generation</run_depend>


  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
```

**CMakeLists.txt**

```
cmake_minimum_required(VERSION 2.8.3)
project(baby_steps)

## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)

## Find catkin macros and libraries
```

```
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  actionlib
  baxter_core_msgs
  baxter_interface
  control_msgs
  cv_bridge
  dynamic_reconfigure
  rospy
  sensor_msgs
  trajectory_msgs
  xacro
)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)


## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
# catkin_python_setup()

################################################
## Declare ROS messages, services and actions ##
################################################

## To declare and build messages, services or actions from within this
## package, follow these steps:
## * Let MSG_DEP_SET be the set of packages whose message types you use in
##   your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
## * In the file package.xml:
##   * add a build_depend tag for "message_generation"
##   * add a build_depend and a run_depend tag for each package in MSG_DEP_SET
##   * If MSG_DEP_SET isn't empty the following dependency has been pulled in
##     but can be declared for certainty nonetheless:
##     * add a run_depend tag for "message_runtime"
## * In this file (CMakeLists.txt):
##   * add "message_generation" and every package in MSG_DEP_SET to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * add "message_runtime" and every package in MSG_DEP_SET to
##     catkin_package(CATKIN_DEPENDS ...)
##   * uncomment the add_*_files sections below as needed
##     and list every .msg/.srv/.action file to be processed
##   * uncomment the generate_messages entry below
##   * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES ...)

## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )

## Generate services in the 'srv' folder
# add_service_files(
#   FILES
#   Service1.srv
#   Service2.srv
# )

## Generate actions in the 'action' folder
add_action_files(
  FILES
  #Action1.action
  Talk.action
)

## Generate added messages and services with any dependencies listed here
```

```
generate_messages(
  DEPENDENCIES
  actionlib_msgs baxter_core_msgs#   control_msgs#   sensor_msgs#   trajectory_msgs
)

################################################
## Declare ROS dynamic reconfigure parameters ##
################################################

## To declare and build dynamic reconfigure parameters within this
## package, follow these steps:
## * In the file package.xml:
##   * add a build_depend and a run_depend tag for "dynamic_reconfigure"
## * In this file (CMakeLists.txt):
##   * add "dynamic_reconfigure" to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * uncomment the "generate_dynamic_reconfigure_options" section below
##     and list every .cfg file to be processed

## Generate dynamic reconfigure parameters in the 'cfg' folder
# generate_dynamic_reconfigure_options(
#   cfg/DynReconf1.cfg
#   cfg/DynReconf2.cfg
# )

###################################
## catkin specific configuration ##
###################################
## The catkin_package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE_DIRS: uncomment this if you package contains header files
## LIBRARIES: libraries you create in this project that dependent projects also need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also need
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES baby_steps
  CATKIN_DEPENDS actionlib baxter_core_msgs baxter_interface control_msgs cv_bridge dynamic_reconfigure rospy sensor_msgs
trajectory_msgs xacro
#  DEPENDS system_lib
)

###########
## Build ##
###########

## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(
# include
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ library
# add_library(${PROJECT_NAME}
#   src/${PROJECT_NAME}/baby_steps.cpp
# )

## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't collide
# add_executable(${PROJECT_NAME}_node src/baby_steps_node.cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following renames the
```

```
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_node"
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Specify libraries to link a library or executable target against
# target_link_libraries(${PROJECT_NAME}_node
#   ${catkin_LIBRARIES}
# )

#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# install(PROGRAMS
#   scripts/my_python_script
#   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark executables and/or libraries for installation
# install(TARGETS ${PROJECT_NAME} ${PROJECT_NAME}_node
#   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
#   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
#   FILES_MATCHING PATTERN "*.h"
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

#############
## Testing ##
#############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_baby_steps.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)
```

**camera.py -**

```
#!/usr/bin/env python

import rospy
import std_srvs.srv

from baxter_core_msgs.srv import(
  ListCameras,
)
```

```
import sys
import baxter_interface

from baxter_interface.camera import CameraController

import subprocess

rospy.init_node("camera", anonymous = True)

"""
It is very important to have the rospy.sleep
inbetween each call to the camera service.
You must allow a minimum of 10 seconds which
is why the sleep was set to 11 seconds.
If not, then a subsequent call to the camera
service is coming from the same node and
you will get an error.
"""

listCameras = 'rosrun baxter_tools camera_control.py -l'
closeLeft = 'rosrun baxter_tools camera_control.py -c left_hand_camera'
closeHead = 'rosrun baxter_tools camera_control.py -c head_camera'
openLeft = 'rosrun baxter_tools camera_control.py -o left_hand_camera -r 960x600'

subprocess.Popen(listCameras, shell=True)
rospy.sleep(11)
subprocess.Popen(closeLeft,shell = True)
rospy.sleep(11)
print("The left hand camera was closed")
subprocess.Popen(closeHead, shell = True)
rospy.sleep(11)
print("The head camera was closed")
subprocess.Popen(openLeft, shell = True)
print("The left hand camera was opened at resolution 960 by 600")
```

**neutral.py -**

```
#!/usr/bin/env python

# Copyright (c) 2013-2015, Rethink Robotics
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are met:
#
# 1. Redistributions of source code must retain the above copyright notice,
#    this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in the
#    documentation and/or other materials provided with the distribution.
# 3. Neither the name of the Rethink Robotics nor the names of its
#    contributors may be used to endorse or promote products derived from
#    this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
# AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
# LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
# INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.

"""
ROS IMPORTS
"""
#you always need to import rospy
import rospy
```

```
#needed to find the path to where the images are stored
import rospkg
import cv;
import cv2;
#cv_bridge is the interface between ROS and openCV.
#here is the web address-- wiki.ros.org/cv_bridge.
# you will find tutorials on how to convert OpenCV images to ROS sensor_msgs/Image messages
import cv_bridge

from sensor_msgs.msg import Image


"""
RETHINK IMPORTS
"""
import baxter_interface

from baxter_interface import CHECK_VERSION
#open cv and cv2 are used to create the images published to Baxter's topic /robot/xdisplay
import cv;
import cv2;
#cv_bridge is the interface between ROS and openCV.
#here is the web address-- wiki.ros.org/cv_bridge.
# you will find tutorials on how to convert OpenCV images to ROS sensor_msgs/Image messages
import cv_bridge

# we have to import the message type used to publish to the topic /robot/xdisplay
from sensor_msgs.msg import Image

# this class resets Baxter arms to neutral, resets the facescreen, makes sure the grippers are
# open and then disables the motors.  Code snippets borrowed from rethink examples
class Reset(object):

    def __init__(self):

        self._left_arm = baxter_interface.limb.Limb("left")
        self._right_arm = baxter_interface.limb.Limb("right")
        self._rp = rospkg.RosPack()
        self._images = (self._rp.get_path('baby_steps') + '/Share/images')
        self.pub = rospy.Publisher('/robot/xdisplay', Image, latch=True, queue_size=10)
        print("Getting robot state... ")
        self._rs = baxter_interface.RobotEnable(CHECK_VERSION)
        self._init_state = self._rs.state().enabled
        print("Enabling robot... ")
        self._rs.enable()
        self.right_gripper = baxter_interface.Gripper("right")
        self.left_gripper = baxter_interface.Gripper("left")


    def set_neutral(self):
        print("Moving to neutral pose...")
        self._left_arm.move_to_neutral()
        self._right_arm.move_to_neutral()


    def reset_facescreen(self):

        print('Resetting facescreen')
        img = cv2.imread(self._images + '/default.png')
        msg = cv_bridge.CvBridge().cv2_to_imgmsg(img, encoding= "bgr8")
        self.pub.publish(msg)

    def grippers_reset(self):
        if self.right_gripper.error():
            print("right gripper error, will reset")
            self.right_gripper.reset()

        if self.left_gripper.error():
            print("left gripper error, will reset")
            self.left_gripper.reset()
```

```
        if not self.left_gripper.calibrated():
            print("\ncalibrating left gripper")
            self.left_gripper.calibrate()

        if not self.right_gripper.calibrated():
            print("\ncalibrating right gripper")
            self.right_gripper.calibrate()
        print("Make sure grippers are open")
        self.right_gripper.open()
        # this just puts ros to sleep for 1 sec
        rospy.sleep(1)
        self.left_gripper.open()


    def disable_motors(self):
        if not self._init_state:
            print("Disabling robot...")
            self._rs.disable()
        else:
            print("Disabling the robot...")
            self._rs.disable()
        return True


def main():

    print("Initializing node...")
    rospy.init_node("reset", anonymous = True)
    reset = Reset()
    reset.set_neutral()
    reset.reset_facescreen()
    reset.grippers_reset()
    reset.disable_motors()


    print("\nBaxter's arms now in neutral position, led display back to rethink log and motors have been disabled.")

if __name__ == '__main__':
    main()
```

**Files Utilized, Modified, and Developed by Connor Desmond**

**Files Utilized, Modified, and Developed by Ryan Cook**

chess_real2.py

```
#!/usr/bin/env python

"""
ROS IMPORTS
"""
#this is a pure Python client library for ROS. It allows programmers using python
# to interface with ROS
import rospy
#cv_bridge is the interface between ROS and openCV.
#here is the web address for the tutorial:
#  http://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython
# you will find tutorials on how to convert OpenCV images to ROS sensor_msgs/Image messages
import cv_bridge
from cv_bridge import CvBridge, CvBridgeError
# read about tf (transform) here: http://wiki.ros.org/tf/Tutorials
# there is a tf2 now....program works fine with tf
# here is the info for tf2 should you decide you want to use it: http://wiki.ros.org/tf2
# The web page also contains links to tutorials.
#I put some formulas to convert back and forth between Euler and Quaternions at the bottom of the page
```

```
import tf
# remember that ROS uses topics and streams the data in messages (msgs).
# this program subscribes to the topic: /cameras/left_hand_camera/image (assuming you are using the left hand camera)
# If you type in cd ros_ws, then get into Baxter's shell by typing ./baxter.sh and then finally type
# rostopic info /cameras/left_hand_camera/image .....it will display that the message type for
# the topic /cameras/left_hand_camera/image is sensor_msgs/Image.
# this is the reason you need to import the sensor_msgs/Image
from sensor_msgs.msg import Image
# this program uses other ROS message types, particulary, when
# the ik service is used.
# Here is a link to the geometry_msgs so that you can see all
# the message types.  These message types are used particularly
# when doing transformations.
# http://wiki.ros.org/geometry_msgs
# if you click on any of the message types at the website, the
# message definition will be displayed.
from geometry_msgs.msg import (
    PoseStamped,
    Pose,
    Point,
    Quaternion,
)
# information on standard message types are found here:
#http://wiki.ros.org/std_msgs
# the header will be used to build pose_stamped msg sent to
# Baxter's ik service which is called
# /ExternalTools/left/PositionKinematicsNode/IKService
# if you were to type in rosservice info /ExternalTools/left/PositionKinematicsNode/IKService
# it would show you that the msg type is baxter_core_msgs/SolvePositionIK
# and that one of the arguments is a pose_stamped msg.  (which needs a header)
# The Empty message is used to ensure that the simulator is ready to go.
# It is from the line of code: rospy.wait_for_message("/robot/sim/started", Empty)
# The simulator has a topic called "/robot/sim/started" which puts out an empty message
#    once it is ready to go. You would not need this for the real robot.
from std_msgs.msg import (
    Header,
    Empty,
)

# common ros msgs including service (srv) http://wiki.ros.org/common_msgs
import std_srvs.srv

#this allows one to query about ros packages
# in this program it is used to find the path to image files which
# are used to reset Baxter's LCD "face" during the shut down process
import rospkg

#These are the gazebo_msgs needed to spawn and delete models
from gazebo_msgs.srv import (
    SpawnModel,
    DeleteModel,
)

import imutils

"""
PYTHON IMPORTS
"""
#the next two imports are for the OpenCV libraries.
"""
This example code uses legacy python code for open CV found here:
https://docs.opencv.org/2.4/modules/core/doc/old_basic_structures.html?highlight=createimage#cv.CreateImage
As you get more familiar with Open CV, you may wish to transition to updated versions. (opencv 3.0)
Just be aware that there may be package dependency as well as compatibility issues with cv_bridge and
move cautiously.
"""
import cv
import cv2
#opencv images are converted to and from Numpy arrays
import numpy
```

```
#python module for basic math functions.  In this program you need it for the sqrt()
# in the function called cBoard_iterate
import math
# python module contains a function that allows the user to map a path
# to a folder.  You will see it just below....os.getenv("HOME") + "/Golf/"
import os

#imports a module that contains a function exit() which
#is used to exit python when there is an exception.
# I commented out the line of code that uses it in the baxter_ik_service method
#   because I did not want the program to exit when an ik solution was not found.
#   For right now, I have it just printing that no solution was found.
import sys

#imports a module which has several string classes and useful functions
import string
from shapedetector import ShapeDetector

import random

# in python mutable objects are lists and dictionaries.
#say you create list1 and then assign list2 = list1.
# if you change list2[0] to another value, you may find that list1[0] has
# also been changed. Please be careful with python mutable objects.
# I have had programs with unexpected results because I was changing data
# without realizing it. Read about it here: http://www.geeksforgeeks.org/copy-python-deep-copy-shallow-copy
import copy

"""
RETHINK IMPORTS
"""

import baxter_interface

from baxter_interface import CHECK_VERSION

#These are baxter's core_msgs that allow use of
#  the IK service
from baxter_core_msgs.srv import (
    SolvePositionIK,
    SolvePositionIKRequest,
)

"""

"""


image_directory = os.getenv("HOME") + "/Golf/"

#End of header with all the imports
#####################################################################


# Locate class
class Locate():
    def __init__(self, arm, distance):
        global image_directory
        self.sd = ShapeDetector()
        # arm ("left" or "right")
        self.limb           = arm
        self._rp = rospkg.RosPack()
        #my package is called test.  You need to change this to the name of your package.
        self._images = (self._rp.get_path('baby_steps') + '/Share/images')
        self.limb_interface = baxter_interface.Limb(self.limb)
        self._joint_names = self.limb_interface.joint_names()
        print("Getting robot state.... ")
        self._rs = baxter_interface.RobotEnable(CHECK_VERSION)
        self._init_state = self._rs.state().enabled
        print("Enabling robot... ")
```

```
self._rs.enable()

if arm == "left":
   self.other_limb = "right"
else:
   self.other_limb = "left"

self.other_limb_interface = baxter_interface.Limb(self.other_limb)

# gripper ("left" or "right")
self.gripper = baxter_interface.Gripper(arm)

# image directory
self.image_dir = image_directory

# flag to control saving of analysis images
# used in canny_it function
self.save_images = True

# this is borrowed from the pick and place demo from rethink for the simulator
self._hover_distance = .07

# required position accuracy in metres
self.cBoard_tolerance = 0.005
self.tray_tolerance = 0.007


# An orientation for gripper fingers to be overhead and parallel to the obj
# this orientation was "borrowed" from the baxter_simulator example pick and place
self.overhead_orientation = Quaternion(
                x=-0.0249590815779,
                y=0.999649402929,
                z=0.00737916180073,
                w=0.00486450832011)

self.starting_right_joint_angles = {'right_w0': 0.38,
                    'right_w1': 1.18,
                    'right_w2': 1.97,
                    'right_e0': .71,
                    'right_e1': 2.42,
                    'right_s0': -0.86,
                    'right_s1': -2.13}


# number of cBoards found
#self.cBoards_found = 0

# start positions
self.cBoard_tray_x = 0.5 #.5                  # x     = front back
self.cBoard_tray_y = 0.3  #.3                 # y     = left right ----positive for y is to your left as you face forward
self.cBoard_tray_z = 0.35                     # z     = up down
self.piece_x = 0.50                  # x     = front back ---- positive for x is forward--negative x is back behind you 0.50
self.piece_y = 0.00                  # y     = left right
self.piece_z = 0.15                  # z     = up down
self.roll        = -1.0 * math.pi           # roll  = horizontal
self.pitch       = 0.0 * math.pi            # pitch = vertical
self.yaw         = 0.0 * math.pi            # yaw   = rotation

self.pose = [self.piece_x, self.piece_y, self.piece_z,     \
          self.overhead_orientation]

# camera parameters (NB. other parameters in open_camera)****the description for how this was calculated is on the website
self.cam_calib    = 0.0025                # meters per pixel at 1 meter .0025
self.cam_x_offset = 0.043                 # camera gripper offset
self.cam_y_offset = -0.084                              #-.01 or -.015
        self.cam_y_offset_close = -.030
self.width        = 960 #640                # Camera resolution
self.height       = 600 #400
```

```
# Hough circle accumulator threshold and minimum radius. These were commented out since Hough circle not being used now.
self.hough_accumulator = 35
self.hough_min_radius  = 10#13
self.hough_max_radius  = 50#35

# canny image-------creates an image header
# parameters: size(image width and height).  Here we are using Baxter's hand camera resolutions.
# next is the bit depth of image elements. Most OpenCV functions use mono8 or bgr8.  Here we use 8.
# Number of channels per pixel.  Most OpenCV functions support 1-4 channels.  We are using 1 channel.
self.canny = cv.CreateImage((self.width, self.height), 8, 1)
"""
Canny uses two thresholds--an upper and lower.  If a pixel gradient is higher than the upper threshold,
the pixel is accepted as an edge.  If a pixel gradient value is below the lower threshold, then it is rejected.
 If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to
 a pixel that is above the upper threshold.
 Recommended upper:lower ration between 2:1 and 3:1
"""
# Canny transform parameters
self.canny_low  = 45
self.canny_high = 150

# minimum cBoard tray area
self.min_area = 20000

# callback image
self.cv_image = cv.CreateImage((self.width, self.height), 8, 3)

# colours
self.white = (255, 255, 255)
self.black = (0, 0, 0)

# cBoard tray corners
self.cBoard_tray_corner = [(0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0)]

# cBoard tray places
# you may choose to use these in the future for the board squares
self.cBoard_tray_place = [(0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                    (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                    (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                    (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                                        (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                    (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                    (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                    (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                                        (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                    (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                    (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                    (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                    (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                                        (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                    (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                    (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                    (0.0,0.0),(0.0,0.0),(0.0,0.0),(0.0,0.0),
                                        (0.0,0.0), (0.0,0.0)]

# Enable the actuators
#baxter_interface.RobotEnable().enable()

# set speed as a ratio of maximum speed -- the default is 0.3.  Use this
#self.limb_interface.set_joint_position_speed(0.5)
#self.other_limb_interface.set_joint_position_speed(0.5)

# create image publisher to head monitor
# enable 'latching on the connect means the last message published is saved and sent to any
# future subscribers that connect.  This is useful for slow-changing or static data like a map.
self.pub = rospy.Publisher('/robot/xdisplay', Image, latch=True, queue_size=10)

# check if gripper error, if yes, reset
# check if calibrated, if not calibrate the gripper
                    # grippers do not open and close unless they are calibrated.
```

```
    if self.gripper.error():
        self.gripper.reset();
        print("There was a gripper error, needed reset")
    if(not self.gripper.calibrated()):
        self.gripper.calibrate()
        print("Calibrated the left gripper")

    # display the start splash screen
    self.splash_screen("Chess", "Let's Play")

    """
    You can access Baxter's two hand cameras and the head camera using the standard ROS image types and image_transport mechanism
    listed below. You can use the ROS Services to open, close, and configure each of the cameras. See the Camera Control Example
    and Using the Cameras for more information on using the cameras. Useful tools for using cameras in ROS include rviz and the
    image_view program. IMPORTANT: You can only have two cameras open at a time at standard resolutions, due to bandwidth limitations.
     The hand cameras are opened by default at boot-time,


    IN THE SIMULATOR, NONE OF THE CAMERA SERVICES WORK--THEREFORE THEY ARE ALL COMMENTED OUT
    In order to have the resolution for the simulator camera at 960 and 600, I had to modify
    baxter_base.gazebo.xacro file. The default is 800 800.  If you go to the "gazebo reference = "left_hand_camera" within
    the document, you will see where you can change the width and height from 800 800 to 960 600 respectively.
    Here is the path to the file:
    home/ros_ws/baxter_common/baxter_description/urdf/baxter_base/baxter_base.gazebo.xacro
    # reset cameras

            """

            #self.reset_cameras()

    # when the left camera closed, the power was sent to the right and head cameras
    #self.close_camera("left")

    #self.close_camera("right")
    #now we close the head camera and the power should go back to the right and
    # left hand camera
    #self.close_camera("head")

    # open required camera...in our case it is the left_hand_camera.  It should open with
    # a resolution of 960, 600.  open_camera is a function in this program.
    #self.open_camera(self.limb, self.width, self.height)

    # subscribe to required camera
    self.subscribe_to_camera(self.limb)

    # distance of arm to table and cBoard tray
    self.distance      = distance
    #this will be the height down to the board once it is made and depending
    # on how thick it is.  Remember distances are in meters.
    self.tray_distance = distance - 0


    # move other arm out of harms way
    if arm == "left":
        self.baxter_ik_move("right", (0.25, -0.50, 0.2, math.pi, 0.0, 0.0))
    else:
        self.baxter_ik_move("left", (0.25, 0.50, 0.2, math.pi, 0.0, 0.0))



def set_neutral(self):
    print("Moving to neutral pose...")
    self.limb_interface.move_to_neutral()
    self.other_limb_interface.move_to_neutral()

def reset(self):

    print('Resetting picture')
    print("Picture reset")
    img = cv2.imread(self._images + '/default.png')
```

```
    msg = cv_bridge.CvBridge().cv2_to_imgmsg(img, encoding= "bgr8")
    self.pub.publish(msg)

def clean_shutdown(self):
    print("\nExiting example...")
    #return to normal
    self.set_neutral()
    rospy.sleep(1)
    self.reset()
    self.gripper.open()

    if not self._init_state:
        print("Disabling robot...")
        self._rs.disable()
    else:
        print("Need to disable robot...")
        self._rs.disable()
    return True

# reset all cameras (incase cameras fail to be recognised on boot)
def reset_cameras(self):
    reset_srv = rospy.ServiceProxy('cameras/reset', std_srvs.srv.Empty)
    rospy.wait_for_service('cameras/reset', timeout=10)
    reset_srv()

# open a camera and set camera parameters
#this function needs rework to update
def open_camera(self, camera, x_res, y_res):
    if camera == "left":
        cam = baxter_interface.camera.CameraController("left_hand_camera")
    elif camera == "right":
        cam = baxter_interface.camera.CameraController("right_hand_camera")
    elif camera == "head":
        cam1 = baxter_interface.camera.CameraController("head_camera")
        cam1.close()
    else:
        sys.exit("ERROR - open_camera - Invalid camera")

    # close camera--comment the next line out because this automatically opens the head camera if
    # you close either the right or left hand camera
    #cam.close()

    # set camera parameters
    cam.resolution          = (int(x_res), int(y_res))
    cam.exposure            = -1            # range, 0-100 auto = -1
    cam.gain                = -1            # range, 0-79 auto = -1
    cam.white_balance_blue  = -1            # range 0-4095, auto = -1
    cam.white_balance_green = -1            # range 0-4095, auto = -1
    cam.white_balance_red   = -1            # range 0-4095, auto = -1

    # open camera
    cam.open()

# close a camera
def close_camera(self, camera):
    if camera == "left":
        cam = baxter_interface.camera.CameraController("left_hand_camera")
    elif camera == "right":
        cam = baxter_interface.camera.CameraController("right_hand_camera")
    elif camera == "head":
        cam = baxter_interface.camera.CameraController("head_camera")
    else:
        sys.exit("ERROR - close_camera - Invalid camera")

    # set camera parameters to automatic
    cam.exposure            = -1            # range, 0-100 auto = -1
    cam.gain                = -1            # range, 0-79 auto = -1
    cam.white_balance_blue  = -1            # range 0-4095, auto = -1
    cam.white_balance_green = -1            # range 0-4095, auto = -1
    cam.white_balance_red   = -1            # range 0-4095, auto = -1
```

```
    # close camera
    cam.close()

# convert Baxter point to image pixel
def baxter_to_pixel(self, pt, dist):
    x = (self.width / 2)                                        \
      + int((pt[1] - (self.pose[1] + self.cam_y_offset)) / (self.cam_calib * dist))
    y = (self.height / 2)                                       \
      + int((pt[0] - (self.pose[0] + self.cam_x_offset)) / (self.cam_calib * dist))

    return (x, y)

# convert image pixel to Baxter point
def pixel_to_baxter(self, px, dist):
    x = ((px[1] - (self.height / 2)) * self.cam_calib * dist) + self.pose[0] + self.cam_x_offset
    y = ((px[0] - (self.width / 2)) * self.cam_calib * dist) + self.pose[1] + self.cam_y_offset

    return (x, y)

# convert image pixel to Baxter point
def pixel_to_baxter_close(self, px, dist):
    x = ((px[1] - (self.height / 2)) * self.cam_calib * dist) + self.pose[0] + self.cam_x_offset
    y = ((px[0] - (self.width / 2)) * self.cam_calib * dist) + self.pose[1] + self.cam_y_offset_close

    return (x, y)

# Not a tree walk due to python recursion limit
def tree_walk(self, image, x_in, y_in):
    almost_black = (1, 1, 1)

    pixel_list = [(x_in, y_in)]                 # first pixel is black save position
    cv.Set2D(image, y_in, x_in, almost_black)     # set pixel to almost black
    to_do = [(x_in, y_in - 1)]                  # add neighbours to to do list
    to_do.append([x_in, y_in + 1])
    to_do.append([x_in - 1, y_in])
    to_do.append([x_in + 1, y_in])

    while len(to_do) > 0:
        x, y = to_do.pop()                      # get next pixel to test
        if cv.Get2D(image, y, x)[0] == self.black[0]:  # if black pixel found
            pixel_list.append([x, y])               # save pixel position
            cv.Set2D(image, y, x, almost_black)       # set pixel to almost black
            to_do.append([x, y - 1])                # add neighbours to to do list
            to_do.append([x, y + 1])
            to_do.append([x - 1, y])
            to_do.append([x + 1, y])

    return pixel_list

# Remove artifacts and find largest object
def look_for_cBoard_tray(self, canny):
    width, height = cv.GetSize(canny)

    centre   = (0, 0)
    max_area = 0

    # for all but edge pixels
    for x in range(1, width - 2):
        for y in range(1, height - 2):
            if cv.Get2D(canny, y, x)[0] == self.black[0]:       # black pixel found
                pixel_list = self.tree_walk(canny, x, y)        # tree walk pixel
                if len(pixel_list) < self.min_area:             # if object too small
                    for l in pixel_list:
                        cv.Set2D(canny, l[1], l[0], self.white) # set pixel to white
                else:                                           # if object found
                    n = len(pixel_list)
                    if n > max_area:                            # if largest object found
                        sum_x  = 0                              # find centre of object
                        sum_y  = 0
```

```
                    for p in pixel_list:
                        sum_x  = sum_x + p[0]
                        sum_y  = sum_y + p[1]

                    centre = sum_x / n, sum_y / n          # save centre of object
                    max_area = n                           # save area of object

        if max_area > 0:                                   # in tray found
            cv.Circle(canny, (centre), 9, (250, 250, 250), -1)      # mark tray centre

        # display the modified canny
        cv.ShowImage("Modified Canny", canny)

        # 3ms wait
        cv.WaitKey(3)
        return centre                                  # return centre of object

    # flood fill edge of image to leave objects
    def flood_fill_edge(self, canny):
        width, height = cv.GetSize(canny)

        #(name of array, row, column, value)
        # set boarder pixels to white
        for x in range(width):
            cv.Set2D(canny, 0, x, self.white)
            cv.Set2D(canny, height - 1, x, self.white)

        for y in range(height):
            cv.Set2D(canny, y, 0, self.white)
            cv.Set2D(canny, y, width - 1, self.white)

        # prime to do list
        to_do = [(2, 2)]
        to_do.append([2, height - 3])
        to_do.append([width - 3, height - 3])
        to_do.append([width - 3, 2])

        while len(to_do) > 0:
            x, y = to_do.pop()                             # get next pixel to test
            if cv.Get2D(canny, y, x)[0] == self.black[0]:    # if black pixel found
                cv.Set2D(canny, y, x, self.white)          # set pixel to white
                to_do.append([x, y - 1])                   # add neighbours to to do list
                to_do.append([x, y + 1])
                to_do.append([x - 1, y])
                to_do.append([x + 1, y])

    # camera call back function
    # consider changing the callback function to only take the data.
    def camera_callback(self, data, camera_name):
        # Convert image from a ROS image message to a CV image
        try:
            # using ROS cv_bridge to convert the image to a CV image.
            # go to the web page: http://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython
            # there is a tutorial there
            # data is the image that is coming from the rostopic via the hand_camera
            self.cv_image = cv_bridge.CvBridge().imgmsg_to_cv2(data, "bgr8")
        except cv_bridge.CvBridgeError, e:
            print e

        # 3ms wait --wait for 3 milliseconds
        cv.WaitKey(3)

    # left camera call back function
    #not sure why we need the second parameter
    # consider changing camera_callback function so that it only takes just the data
    def left_camera_callback(self, data):
        self.camera_callback(data, "Left Hand Camera")

    # right camera call back function
    def right_camera_callback(self, data):
```

```
    self.camera_callback(data, "Right Hand Camera")

# head camera call back function
def head_camera_callback(self, data):
    self.camera_callback(data, "Head Camera")

# create subscriber to the required camera
#this is one of Baxter's topic. When a topic is subscribed to
# there must be a callback function.
def subscribe_to_camera(self, camera):
    if camera == "left":
        callback = self.left_camera_callback
        camera_str = "/cameras/left_hand_camera/image"
    elif camera == "right":
        callback = self.right_camera_callback
        camera_str = "/cameras/right_hand_camera/image"
    elif camera == "head":
        callback = self.head_camera_callback
        camera_str = "/cameras/head_camera/image"
    else:
        sys.exit("ERROR - subscribe_to_camera - Invalid camera")

    camera_sub = rospy.Subscriber(camera_str, Image, callback)

# Convert cv image to a numpy array
def cv2array(self, im):
    depth2dtype = {cv.IPL_DEPTH_8U: 'uint8',
                   cv.IPL_DEPTH_8S: 'int8',
                   cv.IPL_DEPTH_16U: 'uint16',
                   cv.IPL_DEPTH_16S: 'int16',
                   cv.IPL_DEPTH_32S: 'int32',
                   cv.IPL_DEPTH_32F: 'float32',
                   cv.IPL_DEPTH_64F: 'float64'}

    arrdtype=im.depth
    a = numpy.fromstring(im.tostring(),
                         dtype = depth2dtype[im.depth],
                         count = im.width * im.height * im.nChannels)
    a.shape = (im.height, im.width, im.nChannels)

    return a
"""

"""
#the original code imported conversions from moveit commander in the header.
# This caused an error only on shut down---did not affect the program.  Rather than deal with errors
# even at shut down while you are trying to implement code, I went to
# web site below and copied and pasted the two needed function here with
# slight modification only in the except line where you print "Unexpected error"
# https://github.com/ros-planning/moveit/blob/kinetic-devel/moveit_commander/src/moveit_commander/conversions.py
# These functions create the PoseStamped msgs needed for the ik service.
def list_to_pose(self, pose_list):
    pose_msg = Pose()
    try:
        if len(pose_list) == 7:
            pose_msg.position.x = pose_list[0]
            pose_msg.position.y = pose_list[1]
            pose_msg.position.z = pose_list[2]
            pose_msg.orientation.x = pose_list[3]
            pose_msg.orientation.y = pose_list[4]
            pose_msg.orientation.z = pose_list[5]
            pose_msg.orientation.w = pose_list[6]
        elif len(pose_list) == 6:
            pose_msg.position.x = pose_list[0]
            pose_msg.position.y = pose_list[1]
            pose_msg.position.z = pose_list[2]
            q = tf.transformations.quaternion_from_euler(pose_list[3], pose_list[4], pose_list[5])
            pose_msg.orientation.x = q[0]
            pose_msg.orientation.y = q[1]
            pose_msg.orientation.z = q[2]
```

```
            pose_msg.orientation.w = q[3]
        except:
            print "Unexpected error:", sys.exc_info()[0]
            raise
        #uncomment if you want to print the pose message to see what is sent to baxter_ik_move
        #print("**********************")
        #print pose_msg
        return pose_msg

    def list_to_pose_stamped(self, pose_list, target_frame):
        pose_msg = PoseStamped()
        pose_msg.pose = self.list_to_pose(pose_list)
        pose_msg.header.frame_id = target_frame
        pose_msg.header.stamp = rospy.Time.now()
        return pose_msg


    # move a limb
    def baxter_ik_move(self, limb, rpy_pose):
        quaternion_pose = self.list_to_pose_stamped(rpy_pose, "base")
        #quaternion_pose = conversions.list_to_pose_stamped(rpy_pose, "base")
        node = "ExternalTools/" + limb + "/PositionKinematicsNode/IKService"
        ik_service = rospy.ServiceProxy(node, SolvePositionIK)
        ik_request = SolvePositionIKRequest()
        hdr = Header(stamp=rospy.Time.now(), frame_id="base")

        ik_request.pose_stamp.append(quaternion_pose)

        try:
            rospy.wait_for_service(node, 5.0)
            ik_response = ik_service(ik_request)
        except (rospy.ServiceException, rospy.ROSException), error_message:
            rospy.logerr("Service request failed: %r" % (error_message,))
            sys.exit("ERROR - baxter_ik_move - Failed to append pose")

        if ik_response.isValid[0]:
            #self.splash_screen("Valid", "move")
            print("PASS: Valid joint configuration found")
            # convert response to joint position control dictionary
            limb_joints = dict(zip(ik_response.joints[0].name, ik_response.joints[0].position))
            # move limb
            if self.limb == limb:
                self.limb_interface.move_to_joint_positions(limb_joints)
            else:
                self.other_limb_interface.move_to_joint_positions(limb_joints)
        else:
            # display invalid move message on head display
            #self.splash_screen("Invalid", "move")
            # little point in continuing so exit with error message
            print "requested move =", rpy_pose
            print("Requested move was not valid")
            #this will try 3 more times to see if the requested move has a valid solution
            if limb == 'left':
                # requesting the starting position again for this program--will need a better solution
                self.pose = (self.cBoard_tray_x,
                             self.cBoard_tray_y,
                             self.cBoard_tray_z,
                             self.roll, self.pitch, self.yaw)
                self.baxter_ik_move(self.limb, self.pose)
                #return False
                #sys.exit("ERROR - baxter_ik_move - No valid joint configuration found")
            else:
                #self.baxter_ik_move("right", (0.25, -0.50, 0.2, math.pi, 0.0, 0.0))
                print("moving right arm out of the way")
                self.other_limb_interface.move_to_joint_positions(self.starting_right_joint_angles)



        if self.limb == limb:               # if working arm
            """
```

```
        The package baxter_interface is found at api.rethinkrobotics.com/baxter_interface/html/index.html
        In the limb class, the functions are described.
        endpoint_pose() returns the Cartesian endpoint pose {position, orientation}. tf (transform frame) of the gripper
                    All measurements reported for the endpoint(pose, twist, and wrench) are with respect to the /base frame
                    of the robot.
        pose = {'position': (x,y,z), 'orientation': (x,y,z,w)}
        Therefore, for the line of code that says position = quaternion_pose['position'],
        this means that position = [(x,y,z)] of the endpoint_pose
        """
        quaternion_pose = self.limb_interface.endpoint_pose()
        position        = quaternion_pose['position']

        # if working arm remember actual (x,y) position achieved
        # if Baxter's left arm reached the coordinates over the board that we want,
        # we are saving those x and y positions. This is why we are using the x and y
        # from position above.
        self.pose = [position[0], position[1],                          \
                    self.pose[2], self.pose[3], self.pose[4], self.pose[5]]


# update pose in x and y direction
def update_pose(self, dx, dy):
    x = self.pose[0] + dx
    y = self.pose[1] + dy
    pose = [x, y, self.pose[2], self.roll, self.pitch, self.yaw]
    self.baxter_ik_move(self.limb, pose)

# used to place camera over the cBoard tray
def cBoard_tray_iterate(self, iteration, centre):
    # print iteration number
    print "Egg Tray Iteration ", iteration

    # find displacement of object from centre of image
    pixel_dx    = (self.width / 2) - centre[0]
    pixel_dy    = (self.height / 2) - centre[1]
    pixel_error = math.sqrt((pixel_dx * pixel_dx) + (pixel_dy * pixel_dy))
    error       = float(pixel_error * self.cam_calib * self.tray_distance)
    print("Here is the error: ", error)
    print("Here is the self.tray_tolerance: ", self.tray_tolerance)

    x_offset = - pixel_dy * self.cam_calib * self.tray_distance
    y_offset = - pixel_dx * self.cam_calib * self.tray_distance

    # if error in current position too big

    if error > self.tray_tolerance:
        # correct pose
        self.update_pose(x_offset, y_offset)
        # find new centre
        centre = self.canny_it(iteration)

        # find displacement of object from centre of image
        pixel_dx    = (self.width / 2) - centre[0]
        pixel_dy    = (self.height / 2) - centre[1]
        pixel_error = math.sqrt((pixel_dx * pixel_dx) + (pixel_dy * pixel_dy))
        error       = float(pixel_error * self.cam_calib * self.tray_distance)

    return centre, error

# randomly adjust a pose to dither arm position
# used to prevent stalemate when looking for cBoard tray
# the numbers generated will be floats between 0.0 to 1.0
def dither(self):
    x = self.cBoard_tray_x
    y = self.cBoard_tray_y + (random.random() / 10.0)
    pose = (x, y, self.cBoard_tray_z, self.roll, self.pitch, self.yaw)

    return pose

# find the cBoard tray
```

```
def canny_it(self, iteration):
    if self.save_images:
        # save raw image of cBoard tray
        file_name = self.image_dir + "board_" + str(iteration) + ".jpg"
        cv.SaveImage(file_name, cv.fromarray(self.cv_image))
        width, height = cv.GetSize(cv.fromarray(self.cv_image))
        print("Here is the width: {0} and here is the height: {1}.".format(width, height))

    # create an empty image variable, the same dimensions as our camera feed.
    gray = cv.CreateImage((cv.GetSize(cv.fromarray(self.cv_image))), 8, 1)

    # convert the image to a grayscale image
    cv.CvtColor(cv.fromarray(self.cv_image), gray, cv.CV_BGR2GRAY)

    # display image on head monitor
    font     = cv.InitFont(cv.CV_FONT_HERSHEY_SIMPLEX, 1.0, 1.0, 1)
    position = (30, 60)
    cv.PutText(cv.fromarray(self.cv_image), "Looking for Chess Board", position, font, self.white)
    msg = cv_bridge.CvBridge().cv2_to_imgmsg(self.cv_image, encoding="bgr8")
    self.pub.publish(msg)


    #create a Trackbar for user to enter threshold
    #The info can be obtained on github by Atlas7/opencv_python_tutorials.
    # he shows how to create a tracker bar to adjust the greyscale thresholds
    # http://mathalope.co.uk/2015/06/03/canny-edge-detection-app-with-opencv-python/
    # create a canny edge detection map of the greyscale image
    cv.Canny(gray, self.canny, self.canny_low, self.canny_high, 3)

    # display the canny transformation
    cv.ShowImage("Canny Edge Detection", self.canny)

    if self.save_images:
        # save Canny image of cBoard tray
        file_name = self.image_dir + "canny_tray_" + str(iteration) + ".jpg"
        cv.SaveImage(file_name, self.canny)

    # flood fill edge of image to leave only objects
    self.flood_fill_edge(self.canny)
    cBoard_tray_centre = self.look_for_cBoard_tray(self.canny)

    # 3ms wait
    cv.WaitKey(3)

    while cBoard_tray_centre[0] == 0:
        if random.random() > 0.6:
            self.baxter_ik_move(self.limb, self.dither())

        cBoard_tray_centre = self.canny_it(iteration)
        print("Here are the coordinates of cBoard_tray_centre: ", cBoard_tray_centre)

    return cBoard_tray_centre


def find_places(self, c):
    # find long side of cBoard tray
    cc = [c[0], c[1], c[2], c[3]]

    # cBoard tray corners in baxter coordinates
    for i in range(4):
        self.cBoard_tray_corner[i] = self.pixel_to_baxter(cc[i], self.tray_distance)

    # cBoard tray places in pixel coordinates
    ref_x = cc[0][0]
    ref_y = cc[0][1]
    dl_x  = (cc[1][0] - cc[0][0]) / 8
    dl_y  = (cc[1][1] - cc[0][1]) / 8
    ds_x  = (cc[2][0] - cc[1][0]) / 8
    ds_y  = (cc[2][1] - cc[1][1]) / 8
    #please see the diagram with the map of the functions location and how
```

```
    # they corrolate to the colors.  The order below is different than that of
    # the original.
    p     = {}
    tempcount = 0
    for i in range(8):
        for j in range(8):
            p[tempcount] = (ref_x + ((8-i+.7) * dl_x) + ((j+.7) * ds_x), ref_y + ((8-i-.4) * dl_y) + ((j-1-.4) * ds_y))
            tempcount = tempcount + 1

    p[64] = (p[31][0]+100, p[31][1])
            p[65] = (p[28][0]-20, p[28][1] + 20)
    for i in range(66):
        # mark position of cBoard tray places ---different colors were used so you could correlate what the calculations
        #   above are doing.
        #parameters are img, center, radius, circle color(bgr for lime green is (0,250,0)), -1 means that it will fill the shape)
        cv.Circle(cv.fromarray(self.cv_image), (int(p[i][0]), int(p[i][1])), 5, (0, 255, 0), -1)
        self.cBoard_tray_place[i] = self.pixel_to_baxter(p[i], self.tray_distance)


    # display the cBoard tray places
    cv.ShowImage("Egg tray", cv.fromarray(self.cv_image))
    if self.save_images:
        # save cBoard tray image with overlay of cBoard tray and cBoard positions
        file_name = self.image_dir + "chess_board.jpg"
        cv.SaveImage(file_name, cv.fromarray(self.cv_image))

    # 3ms wait
    cv.WaitKey(3)

# find four corners of the cBoard tray
def find_corners(self, centre):
    self.center = centre
    # find bottom corner
    max_x  = 0
    max_y  = 0

    for x in range(100, self.width -100):
    #for x in range(100, self.width - 100):
        y = self.height - 20
        #(array, index1, index2)
        while y > 0 and cv.Get2D(self.canny, y, x)[0] > 100:
        #while y > 0 and cv.Get2D(self.canny, y, x)[0] > 100:
            y = y - 1
        if y > 20:
            cv.Set2D(cv.fromarray(self.cv_image), y, x, (255, 0, 0)) #red is bgr(0,0,255)
            if y > max_y:
                max_x = x
                max_y = y

    corner_1 = (max_x, max_y)

    # find left corner
    min_x  = self.width
    min_y  = 0

    for y in range(100, self.height - 100):
        x = 20
        while x < self.width - 1 and cv.Get2D(self.canny, y, x)[0] > 100:
            x = x + 1
        if x < self.width - 20:
            cv.Set2D(cv.fromarray(self.cv_image), y, x, (0, 255, 0, 0))
            if x < min_x:
                min_x = x
                min_y = y

    corner_2 = (min_x, min_y)

    # display corner image
    cv.ShowImage("Corner", cv.fromarray(self.cv_image))
```

```
    if self.save_images:
        # save Canny image
        file_name = self.image_dir + "board_canny.jpg"
        cv.SaveImage(file_name, self.canny)

        # mark corners and save corner image
        cv.Circle(cv.fromarray(self.cv_image), corner_1, 9, (0, 0, 255), -1)
        cv.Circle(cv.fromarray(self.cv_image), corner_2, 9, (0, 250, 0), -1)
        file_name = self.image_dir + "corner.jpg"
        cv.SaveImage(file_name, cv.fromarray(self.cv_image))

    # 3ms wait
    cv.WaitKey(3)

    # two corners found and centre known find other two corners
    corner_3 = ((2 * centre[0]) - corner_1[0], (2 * centre[1]) - corner_1[1])
    corner_4 = ((2 * centre[0]) - corner_2[0], (2 * centre[1]) - corner_2[1])

    if self.save_images:
        # save Canny image
        file_name = self.image_dir + "board_canny.jpg"
        cv.SaveImage(file_name, self.canny)

        # mark corners and save corner image
        cv.Circle(cv.fromarray(self.cv_image), corner_1, 9, (0, 0, 255), -1)
        cv.Circle(cv.fromarray(self.cv_image), corner_2, 9, (0, 250, 0), -1)
        cv.Circle(cv.fromarray(self.cv_image), corner_3, 9, (255, 0, 0), -1)
        cv.Circle(cv.fromarray(self.cv_image), corner_4, 9, (255, 250, 0), -1)

        file_name = self.image_dir + "all_four_corners.jpg"
        cv.SaveImage(file_name, cv.fromarray(self.cv_image))

    # draw cBoard tray boundry
    c1 = (int(corner_1[0]), int(corner_1[1]))
    c2 = (int(corner_2[0]), int(corner_2[1]))
    c3 = (int(corner_3[0]), int(corner_3[1]))
    c4 = (int(corner_4[0]), int(corner_4[1]))

    cv.Line(cv.fromarray(self.cv_image), c1, c2, (255, 0, 0), thickness=3)
    cv.Line(cv.fromarray(self.cv_image), c2, c3, (0, 255, 0), thickness=3)
    cv.Line(cv.fromarray(self.cv_image), c3, c4, (255, 255, 0), thickness=3)
    cv.Line(cv.fromarray(self.cv_image), c4, c1, (0, 0, 255), thickness=3)

    return True, (corner_1, corner_2, corner_3, corner_4)

# find the cBoard tray
def find_cBoard_tray(self):
    #define and intialize boolean variable named ok
    ok = False
    #while ok is false
    while not ok:
        cBoard_tray_centre = self.canny_it(0)

        error     = 2 * self.tray_tolerance
        iteration = 1

        # iterate until arm over centre of tray
        while error > self.tray_tolerance:
            cBoard_tray_centre, error = self.cBoard_tray_iterate(iteration,       \
                                cBoard_tray_centre)
            iteration             += 1

        # find cBoard tray corners in pixel units
        (ok, corners) = self.find_corners(cBoard_tray_centre)

    self.find_places(corners)

# display message on head display
def splash_screen(self, s1, s2):
    splash_array = numpy.zeros((self.height, self.width, 3), numpy.uint8)
```

```
    font = cv.InitFont(cv.CV_FONT_HERSHEY_SIMPLEX, 3.0, 3.0, 9)

    ((text_x, text_y), baseline) = cv.GetTextSize(s1, font)
    org = ((self.width - text_x) / 2, (self.height / 3) + (text_y / 2))
    cv2.putText(splash_array, s1, org, cv.CV_FONT_HERSHEY_SIMPLEX, 3.0,         \
            self.white, thickness = 7)

    ((text_x, text_y), baseline) = cv.GetTextSize(s2, font)
    org = ((self.width - text_x) / 2, ((2 * self.height) / 3) + (text_y / 2))
    cv2.putText(splash_array, s2, org, cv.CV_FONT_HERSHEY_SIMPLEX, 3.0,         \
            self.white, thickness = 7)

    splash_image = cv.fromarray(splash_array)

    # 3ms wait
    cv2.waitKey(3)

    msg = cv_bridge.CvBridge().cv2_to_imgmsg(splash_array, encoding="bgr8")
    self.pub.publish(msg)



def find_offset_pose(self, pose):
    pose1 = copy.deepcopy(pose)
    print("Here is the pose coming into find_offset_pose: ", pose1)
    list_of_circles = {}
    file_name = self.image_dir + "circles" + ".jpg"
    cv.SaveImage(file_name, cv.fromarray(self.cv_image))
    image = cv2.imread(file_name)
    cv2.imshow("Test Circle", image)
    #cv2.waitKey(0)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5,5),0)
    thresh = cv2.threshold(blurred,40, 255, cv2.THRESH_BINARY)[1]
    #cv2.imshow("Threshold Picture", thresh)
    thresh = cv2.bitwise_not(thresh)
    #cv2.imshow("Threshold Inverted", thresh)

    #find contours in the threshold image
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
       cv2.CHAIN_APPROX_SIMPLE)
    cnts = cnts[0] if imutils.is_cv2() else cnts[1]

    circle_centers = []
    closest_point = (0,0)
    distance = 10000.00
    length = len(cnts)
    print("The length of cnts is: ", length)
    print("\n" * 2)
    # loop over the contours
    for c in cnts: #[0:6]:
            # compute the center of the contour
       point = (0,0)
            M = cv2.moments(c)

            #print("The moments")
            #print(M)
            if M["m00"] == 0:
          print("******************")
          print("M[m00] equaled 0")
          print("******************")
       else:
                cX = int(M["m10"] / M["m00"])
                cY = int(M["m01"] / M["m00"])
          #shape = self.sd.detect(c)
          #print("Here is the value of shape: ", shape)
          center = str(cX) + ", " + str(cY)
          point = cX, cY
          print("Here is the value of point: ", point)
          #convert to baxter coordinates.
```

```
            cBoard = self.pixel_to_baxter_close(point, .153)
            #find distance between center where contour found and desired pose.
                print("Before Distance calculation")
            distance_new = ((cBoard[0] - pose[0]) * (cBoard[0] - pose[0])) + ((cBoard[1] - pose[1]) * (cBoard[1] - pose[1]))
                print("After Distance Calculation")
            if distance_new < distance:
               distance = distance_new
               closest_point = cBoard[0], cBoard[1]
            center = str(cBoard[0]) + ", " + str(cBoard[1])
            print("Here is the value of cBoard after change to pixel_to_baxter: ", cBoard)
            circle_centers.append(cBoard)
            print("Here is the value of circle_centers in the for-loop: ", circle_centers)
            print('\n' * 2)
                # draw the contour and center of the shape on the image
                cv2.drawContours(image, [c], -1, (0, 255, 0), 2)
                cv2.circle(image, (cX, cY), 7, (255, 255, 255), -1)
                #cv2.putText(image, center, (cX - 20, cY - 20),
                    #cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

                # show the image
                cv2.imshow("Image", image)
                #cv2.waitKey(0)
    print("Here is the value of the closest point outside the for-loop: ", closest_point)
    #calculate the offsets from where you want to go to where the actual contour is located.


            pose1[0] = closest_point[0]
            pose1[1] = closest_point[1]

    print("The updated version of pose1: ", pose1)
    return pose1


def gripper_open(self):
    self.gripper.open()
    rospy.sleep(1.0)

def gripper_close(self):
    self.gripper.close()
    rospy.sleep(1.0)


def _approach(self, pose):
    pose1 = copy.deepcopy(pose)
    pose1[2] = pose1[2] + self._hover_distance

    # approach with a pose the hover-distance above the requested pose
    self.baxter_ik_move(self.limb, pose1)

def _retract(self):
    # retrieve current pose from endpoint
    current_pose = self.limb_interface.endpoint_pose()
    ik_pose = []
    ik_pose.append(current_pose['position'].x)
    ik_pose.append(current_pose['position'].y)
    ik_pose.append(current_pose['position'].z + self._hover_distance)
    ik_pose.append(current_pose['orientation'].x)
    ik_pose.append(current_pose['orientation'].y)
    ik_pose.append(current_pose['orientation'].z)
    ik_pose.append(current_pose['orientation'].w)
    self.baxter_ik_move(self.limb, ik_pose)
    # servo up from current pose


def _servo_to_pose(self, pose):
    # servo down to release
    self.baxter_ik_move(self.limb, pose)


def pick(self, pose):
```

```
        # open the gripper
        self.gripper_open()
        # servo above pose
        self._approach(pose)

                pose_offset = self.find_offset_pose(pose)
                # servo above pose
        self._approach(pose_offset)
        # servo to pose
        self._servo_to_pose(pose_offset)
        # close gripper
        self.gripper_close()
        # retract to clear object
        self._retract()

    def middle(self,pose):
        # servo above pose
        self._approach(pose)

    def place(self, pose):
        # servo above pose
        self._approach(pose)
        # servo to pose
        self._servo_to_pose(pose)
        # open the gripper
        self.gripper_open()
        # retract to clear object
        self._retract()

#END OF CLASS LOCATOR
##########################################################

# read the setup parameters from setup.dat
#This is the first function to look at.It is entirely
# optional.  The limb and distance can be "hard coded" or
#you can even get user input if desired.
def get_setup():
    global image_directory
    file_name = image_directory + "setup.dat"

    try:
        f = open(file_name, "r")
    except ValueError:
        sys.exit("ERROR: setup must be run before this")

    # find limb
    s = string.split(f.readline())
    if len(s) >= 3:
        if s[2] == "left" or s[2] == "right":
            limb = s[2]
        else:
            sys.exit("ERROR: invalid limb in %s" % file_name)
    else:
        sys.exit("ERROR: missing limb in %s" % file_name)

    # find distance to table
    s = string.split(f.readline())
    if len(s) >= 3:
        try:
            distance = float(s[2])
        except ValueError:
            sys.exit("ERROR: invalid distance in %s" % file_name)
    else:
        sys.exit("ERROR: missing distance in %s" % file_name)

    return limb, distance

#this function loads the models used in gazebo
def load_gazebo_models(table_pose=Pose(position=Point(x= .75, y=0.2, z=0.0)),
                       table_reference_frame="world",
```

```
                block_pose=Pose(position=Point(x=0.6, y=0.25, z=0.7825)),
                block_reference_frame="world"):
    # Get Models' Path
    model_path = rospkg.RosPack().get_path('baby_steps')+"/models/"
    # Load Table SDF
    table_xml = ''
    with open (model_path + "cafe_table/model.sdf", "r") as table_file:
        table_xml=table_file.read().replace('\n', '')
    # Load Block URDF
    block_xml = ''
    with open (model_path + "block/chessboard.sdf", "r") as block_file:
        block_xml=block_file.read().replace('\n', '')
    # Spawn Table SDF
    rospy.wait_for_service('/gazebo/spawn_sdf_model')
    try:
        spawn_sdf = rospy.ServiceProxy('/gazebo/spawn_sdf_model', SpawnModel)
        resp_sdf = spawn_sdf("cafe_table", table_xml, "/",
                             table_pose, table_reference_frame)
    except rospy.ServiceException, e:
        rospy.logerr("Spawn SDF service call failed: {0}".format(e))
    # Spawn Block URDF
    rospy.wait_for_service('/gazebo/spawn_urdf_model')
    try:
        spawn_urdf = rospy.ServiceProxy('/gazebo/spawn_sdf_model', SpawnModel)
        resp_urdf = spawn_urdf("block", block_xml, "/",
                               block_pose, block_reference_frame)
    except rospy.ServiceException, e:
        rospy.logerr("Spawn URDF service call failed: {0}".format(e))

def delete_gazebo_models():
    # This will be called on ROS Exit, deleting Gazebo models
    # Do not wait for the Gazebo Delete Model service, since
    # Gazebo should already be running. If the service is not
    # available since Gazebo has been killed, it is fine to error out
    try:
        delete_model = rospy.ServiceProxy('/gazebo/delete_model', DeleteModel)
        resp_delete = delete_model("cafe_table")
        resp_delete = delete_model("block")
    except rospy.ServiceException, e:
        rospy.loginfo("Delete Model service call failed: {0}".format(e))
```

**Files Utilized, Modified, and Developed by Zephaniah Connell**

**pyttsx_server.py -**

```
#! /usr/bin/env python
#Please see the license.txt file for the open source licenses for these two references
"""
References:
"Programming Robots with ROS" by Morgan Quigley, Brian Gerkey, and William Smart.  Chapter 19 is devoted to making the robot talk using
  pyttsx. Pyttsx is text to speech. Code was used from the book.  The code for the book is available at (i) and the documentation for
  pyttsx is available at (ii)
  (i)  https://github.com/osrf/rosbook
  (ii) https://pyttsx.readthedocs.io/en/latest/

Reference for implementing a simple action server is from the ROS Wiki Tutorial Web site:
http://wiki.ros.org/actionlib_tutorials/Tutorials/Writing%20a%20Simple%20Action%20Server%20using%20the%20Execute%20Callback%20%
28Python%29


"""
import rospy
import threading, time, pyttsx
import actionlib
from baby_steps.msg import TalkAction, TalkGoal, TalkResult

class TalkNode():

    def __init__(self, node_name, action_name):
        rospy.init_node(node_name, anonymous = True)
```

```
        #first create a simple action server. Arguments are a name, an action,
        # a call_back, and set auto_start to False
        self.server = actionlib.SimpleActionServer(action_name, TalkAction,
          self.do_talk, False)
        #get a reference to an engine reference from the pyttsx library
        self.engine = pyttsx.init()
        """
        In the next section, we declare a variable used to set the properties for
        the robot voice we will be setting the rate which is integer speech rate in
        words per minute.
        voices is a list of the available "voices" in pyttsx.  We will be using
        id 16.
        We also start a thread to run the speech engine loop
        """
        rate = self.engine.getProperty('rate')
        voices = self.engine.getProperty('voices')
        self.engine_thread = threading.Thread(target=self.loop)
        self.engine_thread.start()
        self.engine.setProperty('volume', rospy.get_param('~volume', 1.0))
        self.engine.setProperty('rate', rate - 90)
        self.engine.setProperty('voice', voices[16].id)
        self.preempt = rospy.get_param('~preempt', False)
        #start the action server
        self.server.start()

    def loop(self):
        self.engine.startLoop(False)
        while not rospy.is_shutdown():
            #when using an external event loop, engine.iterate must
            # be used within the external loop per pyttsx docs
            self.engine.iterate()
            time.sleep(0.1)
        self.engine.endLoop()

    def do_talk(self, goal):
        #when the server receives a goal to speak, it passes it to the speech engine
        # we created.
        self.engine.say(goal.sentence)
        while self.engine.isBusy():
            # action servers allow the action to be preempted. If a
            # request is received then the action would be stopped by using
            # the speech libraries stop function for the engine we created.
            if self.preempt and self.server.is_preempt_requested():
                self.engine.stop()
                while self.engine.isBusy():
                    time.sleep(0.1)
                self.server.set_preempted(TalkResult(), "Talk preempted")
                return
            time.sleep(0.1)
        self.server.set_succeeded(TalkResult(), "Talk completed successfully")

#create an instance of the TalkNode class
talker = TalkNode('speaker', 'speak')
#keep the server spinning
rospy.spin()
```

**pyttsx_client.py -**

```
#! /usr/bin/env python
#Please see the license.txt file for the open source licenses for these two references
"""
References:
"Programming Robots with ROS" by Morgan Quigley, Brian Gerkey, and William Smart.  Chapter 19 is devoted to making the robot talk using
  pyttsx. Pyttsx is text to speech. Code was used from the book.  The code for the book is available at (i) and the documentation for
  pyttsx is available at (ii)
  (i)  https://github.com/osrf/rosbook
  (ii) https://pyttsx.readthedocs.io/en/latest/

Reference to create a simple action client:
http://wiki.ros.org/actionlib_tutorials/Tutorials/Writing%20a%20Simple%20Action%20Client%20%28Python%29
```

```
"""
import rospy

import actionlib
from baby_steps.msg import TalkAction, TalkGoal, TalkResult

class Talk():

    def say_something(self, what_to_say):
            #rospy.init_node('speaker_client')
        #create the client by passing in as arguments the name and type of action
        #in this case, we are using the TalkAction.action that we created
            client = actionlib.SimpleActionClient('speak', TalkAction)
        #we wait until the action server has started and is accepting goals
            client.wait_for_server()
        #TalkGoal was created when we wrote the Talk.action file and then compiled it
        # In Talk.action, we define the goal as string sentence. Thus the data type
        # is a string and the goal is sentence
            goal = TalkGoal()
            goal.sentence = what_to_say
        #send the goal to the action server. In this case, the goal is to say our sentence.
            client.send_goal(goal)
        #wait for the server to give us our results.
            client.wait_for_result()
            print('[Result] State: %d'%(client.get_state()))
            print('[Result] Status: %s'%(client.get_goal_status_text()))


if __name__ =="__main__":
    try:
        rospy.init_node('speaker_client', anonymous = True)
        talk1 = Talk()
        talk1.talk_client()
        #talk1=talk_client()
    except rospy.ROSInterruptException:
        print "program interrupted before completion"
```

**voicecommands.py -**

```
#!/usr/bin/env python


import baxter_interface
from baxter_interface import CHECK_VERSION
import rospy
from std_msgs.msg import (
    String,
    Float64,
)

import rospkg

#import and set up virtual chess logic classes
from Game import Game
game = Game()
game.initial_board.initialize_board()
game.create_piece_dict()
game.print_board()

#import and initialization for board and piece recognition and movement
from chess_real2 import Locate
import string
import copy
import os
image_directory = os.getenv("HOME") + "/Golf/"

#import class for clean shutdown
from neutral import Reset

#import class for text to speech
```

```
from pyttsx_client import Talk

class Voice:
    def __init__(self):

        rs = baxter_interface.RobotEnable(CHECK_VERSION)
        print("Enabling robot... ")
        rs.enable()

        self.rate = 5
        self.r = rospy.Rate(self.rate)

        self.message = 'nothing yet'

        # A flag to determine whether or not voice control is paused
        self.paused = True

                # A flag to determine if the given command is the pickup index or placement index
        self.movementdefined = False

                # variables for the starting and ending positions
        self.piecetomove = 'none'
        self.destination = 'nowhere'

        # variable to determine whose turn it is
        self.whiteturn = True

        # variables to determine if Baxter successfully executed a move if valid
                self.checkedmove = False
        self.waitingforcheck = False
        self.movesuccess = False

        # variable to determine if a chess piece movement has been successfully requested
        self.movementdefined = False

        # Subscribe to the /recognizer/output topic to receive voice commands.
        rospy.Subscriber('/recognizer/output', String, self.speech_callback)

        # Object to be used for text to speech
        self.talk = Talk()


    def speech_callback(self, msg):

        # Get the motion command from the recognized phrase
        command = msg.data

        # Log the command to the screen
        rospy.loginfo("Command: " + str(command))

        # If the user has asked to pause/continue voice control,
        # set the flag accordingly
        if command == 'pause':
            self.paused = True
            what_to_say = "Voice command features disabled."
            self.talk.say_something(what_to_say)
            print(what_to_say)
            return
        elif command == 'continue':
            self.paused = False
            what_to_say = "Voice command features enabled."
            self.talk.say_something(what_to_say)
            print(what_to_say)
            return

        # If voice control is paused, simply return without
        # performing any action
        if self.paused:
            what_to_say = "Currently paused"
            #self.talk.say_something(what_to_say)
```

```
            print(what_to_say)
            return

        if command == 'shutdown':
            #reset = Reset()
            #reset.set_neutral()
            #reset.reset_facescreen()
            #reset.grippers_reset()
            #reset.disable_motors()
            rospy.signal_shutdown('Quit')
            what_to_say = "\nBaxter's arms now in neutral position, led display back to rethink log and motors have been disabled."
            self.talk.say_something(what_to_say)
            print(what_to_say)
            return

        # wait for an answer to whether Baxter successfully made a move or not before accepting any new movements
        if self.movementdefined and not self.checkedmove:
            if command == 'success':
                self.checkedmove = True
                self.movesuccess = True
            elif command == 'failure':
                self.checkedmove = True
                self.movesuccess = False
            else:
                self.paused = True
                what_to_say = "Must answer before requesting another move.\nDid Baxter successfully make the move most recently
requested?\nReply 'Success' or 'Failure'."
                self.talk.say_something(what_to_say)
                print(what_to_say)
                self.paused = False
            return

        piece = 'none'
        index = 'nowhere'

        # get index of piece to move or index to move to
        if command == 'alpha one':
            index = 'a1'
        elif command == 'alpha two':
            index = 'a2'
        elif command == 'alpha three':
            index = 'a3'
        elif command == 'alpha four':
            index = 'a4'
        elif command == 'alpha five':
            index = 'a5'
        elif command == 'alpha six':
            index = 'a6'
        elif command == 'alpha seven':
            index = 'a7'
        elif command == 'alpha eight':
            index = 'a8'
        elif command == 'bravo one':
            index = 'b1'
        elif command == 'bravo two':
            index = 'b2'
        elif command == 'bravo three':
            index = 'b3'
        elif command == 'bravo four':
            index = 'b4'
        elif command == 'bravo five':
            index = 'b5'
        elif command == 'bravo six':
            index = 'b6'
        elif command == 'bravo seven':
            index = 'b7'
        elif command == 'bravo eight':
            index = 'b8'
        elif command == 'charlie one':
            index = 'c1'
```

```
elif command == 'charlie two':
    index = 'c2'
elif command == 'charlie three':
    index = 'c3'
elif command == 'charlie four':
    index = 'c4'
elif command == 'charlie five':
    index = 'c5'
elif command == 'charlie six':
    index = 'c6'
elif command == 'charlie seven':
    index = 'c7'
elif command == 'charlie eight':
    index = 'c8'
elif command == 'delta one':
    index = 'd1'
elif command == 'delta two':
    index = 'd2'
elif command == 'delta three':
    index = 'd3'
elif command == 'delta four':
    index = 'd4'
elif command == 'delta five':
    index = 'd5'
elif command == 'delta six':
    index = 'd6'
elif command == 'delta seven':
    index = 'd7'
elif command == 'delta eight':
    index = 'd8'
elif command == 'echo one':
    index = 'e1'
elif command == 'echo two':
    index = 'e2'
elif command == 'echo three':
    index = 'e3'
elif command == 'echo four':
    index = 'e4'
elif command == 'echo five':
    index = 'e5'
elif command == 'echo six':
    index = 'e6'
elif command == 'echo seven':
    index = 'e7'
elif command == 'echo eight':
    index = 'e8'
elif command == 'foxtrot one':
    index = 'f1'
elif command == 'foxtrot two':
    index = 'f2'
elif command == 'foxtrot three':
    index = 'f3'
elif command == 'foxtrot four':
    index = 'f4'
elif command == 'foxtrot five':
    index = 'f5'
elif command == 'foxtrot six':
    index = 'f6'
elif command == 'foxtrot seven':
    index = 'f7'
elif command == 'foxtrot eight':
    index = 'f8'
elif command == 'golf one':
    index = 'g1'
elif command == 'golf two':
    index = 'g2'
elif command == 'golf three':
    index = 'g3'
elif command == 'golf four':
    index = 'g4'
```

```
elif command == 'golf five':
    index = 'g5'
elif command == 'golf six':
    index = 'g6'
elif command == 'golf seven':
    index = 'g7'
elif command == 'golf eight':
    index = 'g8'
elif command == 'hotel one':
    index = 'h1'
elif command == 'hotel two':
    index = 'h2'
elif command == 'hotel three':
    index = 'h3'
elif command == 'hotel four':
    index = 'h4'
elif command == 'hotel five':
    index = 'h5'
elif command == 'hotel six':
    index = 'h6'
elif command == 'hotel seven':
    index = 'h7'
elif command == 'hotel eight':
    index = 'h8'
elif command == 'pawn one':
    piece = 'P1'
elif command == 'pawn two':
    piece = 'P2'
elif command == 'pawn three':
    piece = 'P3'
elif command == 'pawn four':
    piece = 'P4'
elif command == 'pawn five':
    piece = 'P5'
elif command == 'pawn six':
    piece = 'P6'
elif command == 'pawn seven':
    piece = 'P7'
elif command == 'pawn eight':
    piece = 'P8'
elif command == 'rook one':
    piece = 'R1'
elif command == 'rook two':
    piece = 'R2'
elif command == 'knight one':
    piece = 'N1'
elif command == 'knight two':
    piece = 'N2'
elif command == 'bishop one':
    piece = 'B1'
elif command == 'bishop two':
    piece = 'B2'
elif command == 'king':
    piece = 'K'
elif command == 'queen':
    piece = 'Q'
elif command == 'reset':
    self.piecetomove = 'none'
    self.destination = 'nowhere'
    self.movementdefined = False
    what_to_say = "Reset confirmed. Say the piece you would like to move, or say a functional operation."
    self.talk.say_something(what_to_say)
    print(what_to_say)

else:
    print("No valid command")
    return

if self.piecetomove == 'none' and piece != 'none':
    self.piecetomove = piece
```

```
            what_to_say = self.piecetomove + " has been chosen as the piece to move. Give the location to which you would like to move it."
            self.talk.say_something(what_to_say)
            print(what_to_say)
        if self.destination == 'nowhere' and index != 'nowhere' and self.piecetomove != 'none':
            self.destination = index
            self.movementdefined = True
            what_to_say = self.destination + " has been chosen as the destination."
            self.talk.say_something(what_to_say)
            print(what_to_say)


# read the setup parameters from setup.dat
#This is the first function to look at.It is entirely
# optional.  The limb and distance can be "hard coded" or
#you can even get user input if desired.
def get_setup():
    global image_directory
    file_name = image_directory + "setup.dat"

    try:
        f = open(file_name, "r")
    except ValueError:
        sys.exit("ERROR: setup must be run before this")

    # find limb
    s = string.split(f.readline())
    if len(s) >= 3:
        if s[2] == "left" or s[2] == "right":
            limb = s[2]
        else:
            sys.exit("ERROR: invalid limb in %s" % file_name)
    else:
        sys.exit("ERROR: missing limb in %s" % file_name)

    # find distance to table
    s = string.split(f.readline())
    if len(s) >= 3:
        try:
            distance = float(s[2])
        except ValueError:
            sys.exit("ERROR: invalid distance in %s" % file_name)
    else:
        sys.exit("ERROR: missing distance in %s" % file_name)

    return limb, distance

if __name__=="__main__":

    try:
        rospy.init_node('voicecommands', anonymous = True)

        # get setup parameters---go to function with #2 by it
        limb, distance = get_setup()
        print "limb     = ", limb
        print "distance = ", distance
        # create an instance of the class Locate.
        locator = Locate(limb, distance)
        rospy.on_shutdown(locator.clean_shutdown)
        voice = Voice()
        what_to_say = "Press Enter to start initialization: "
        voice.talk.say_something(what_to_say)
        raw_input("Press Enter to start initialization: ")

        locator.gripper.open()


        # move the arm whose camera you are using close to the board
        # you may wish to change this starting location.
        locator.pose = [locator.cBoard_tray_x,
                        locator.cBoard_tray_y,
```

```
            locator.cBoard_tray_z,
            locator.roll, locator.pitch, locator.yaw]
locator.baxter_ik_move(locator.limb, locator.pose)

# find the chess board
locator.find_cBoard_tray()

# create a list of poses in Baxter's coordinates for each chess board square
board_spot = list()
what_to_say = "Set up board pieces. Then press Enter to begin voice command operation: "
voice.talk.say_something(what_to_say)
raw_input("Set up board pieces. Then press Enter to begin voice command operation: ")
for i in range (66):
            if i >= 64:
                        locator.pose = [copy.copy(locator.cBoard_tray_place[i][0]), #-.015
         copy.copy(locator.cBoard_tray_place[i][1]), #-.045
         locator.piece_z - .15,
         locator.roll,
       locator.pitch,
       locator.yaw]
            else:
       locator.pose = [copy.copy(locator.cBoard_tray_place[i][0]), #-.015
            copy.copy(locator.cBoard_tray_place[i][1]), #-.045
            locator.piece_z - .32,
            locator.roll,
          locator.pitch,
          locator.yaw]
   board_spot.append(locator.pose)
#locator.baxter_ik_move(locator.limb, locator.pose)

#unpause to enable voice command operation
voice.paused = False

what_to_say = "Say the piece you would like to move, or say a functional operation."
voice.talk.say_something(what_to_say)
print(what_to_say)

while not rospy.is_shutdown():
   # if the user defined a possible move attempt to perform it once
   if voice.movementdefined and not voice.checkedmove and not voice.waitingforcheck:
      #print the desired move to the terminal for reference
      print("Attempting to move " + voice.piecetomove + " to postion " + voice.destination)

      #get the coordinates of the starting and ending positions for the desired move
      loc = game.label_to_loc(voice.destination)

      #check if the move is valid using the virtual board
      validmove, piecekilled = game.check_move(voice.piecetomove,loc,voice.whiteturn)

      #request baxter to perform the physical movement
      if validmove:
         #get index of the piece the player wants to move
         if(voice.whiteturn):
            pieceindex = game.white_piece_dict[voice.piecetomove]
         else:
            pieceindex = game.black_piece_dict[voice.piecetomove]

         #get the coordinates of the starting and ending positions for the desired move
         pos1 = 8*(7-pieceindex[1]) + pieceindex[0]
         pos2 = 8*(7-loc[1]) + loc[0]

         voice.paused = True
         if piecekilled == True:
            print("\nKilling")
            locator.pick(board_spot[pos2])
            locator.middle(board_spot[65])
            locator.place(board_spot[64])
            locator.middle(board_spot[65])

         print("\nPicking...")
```

```
                locator.pick(board_spot[pos1])
                locator.middle(board_spot[65])
                print("\nPlacing...")
                updatedpose = copy.copy(board_spot[pos2])
                updatedpose[2] = board_spot[pos2][2] + .01
                locator.place(updatedpose)
                            locator.middle(board_spot[65])
                voice.waitingforcheck = True
                what_to_say = "Did Baxter successfully make the move requested? \nReply 'Success' or 'Failure'."
                voice.talk.say_something(what_to_say)
                print(what_to_say)
                voice.paused = False
            else:
                what_to_say = "Requested move was invalid.\nSay the piece you would like to move, or say a functional operation."
                voice.talk.say_something(what_to_say)
                print(what_to_say)
                voice.movementdefined = False
                voice.piecetomove = 'none'
                voice.destination = 'nowhere'

        winner = False
        # if baxter successfully executed the move
        # update the virtual board state,
        # change turns,
        # and reset variables to be ready for next verbal command
        if voice.movementdefined and voice.checkedmove:
            if voice.movesuccess:
                validmove = game.move_piece(voice.piecetomove,voice.destination,voice.whiteturn) # update virtual board state
                if not game.check_king_state(voice.whiteturn):
                    if voice.whiteturn:
                        what_to_say = "Red player wins."
                        voice.talk.say_something(what_to_say)
                        print(what_to_say)
                    else:
                        what_to_say = "Black player wins."
                        voice.talk.say_something(what_to_say)
                        print(what_to_say)
                    rospy.signal_shutdown('Quit')
                    break
                voice.whiteturn = not voice.whiteturn # change turn between white and black
            game.print_board()
            voice.piecetomove = 'none'
            voice.destination = 'nowhere'
            voice.movementdefined = False
            voice.waitingforcheck = False
            voice.checkedmove = False
            what_to_say = "Say the piece you would like to move, or say a functional operation."
            voice.talk.say_something(what_to_say)
            print(what_to_say)

        voice.r.sleep()

    except rospy.ROSInterruptException:
        rospy.loginfo("Voice command script terminated.")
        reset = Reset()
        reset.set_neutral()
        reset.reset_facescreen()
        reset.grippers_reset()
        reset.disable_motors()
        print("\nBaxter's arms now in neutral position, led display back to rethink log and motors have been disabled.")
```

**speech_commands.launch -**

```
<launch>

  <node name="recognizer" pkg="pocketsphinx" type="recognizer.py" output="screen">
    <param name="lm" value="$(find baby_steps)/config/speech_commands.lm"/>
    <param name="dict" value="$(find baby_steps)/config/speech_commands.dic"/>
  </node>
```

</launch>

**speech_commands.txt -**

one
two
three
four
five
six
seven
eight
alpha
bravo
charlie
delta
echo
foxtrot
golf
hotel
pawn
rook
knight
king
queen
bishop
pause
continue
shutdown
reset
success
failure

**speech_commands.dic -**

ALPHA AE L F AH
BISHOP B IH SH AH P
BRAVO B R AA V OW
CHARLIE CH AA R L IY
CONTINUE K AH N T IH N Y UW
DELTA D EH L T AH
ECHO EH K OW
EIGHT EY T
FAILURE F EY L Y ER
FIVE F AY V
FOUR F AO R
FOXTROT F AA K S T R AA T
GOLF G AA L F
GOLF(2) G AO L F
HOTEL HH OW T EH L
KING K IH NG
KNIGHT N AY T
ONE W AH N
ONE(2) HH W AH N
PAUSE P AO Z
PAWN P AO N
QUEEN K W IY N
RESET R IY S EH T
ROOK R UH K
SEVEN S EH V AH N
SHUTDOWN SH AH T D AW N
SIX S IH K S
SUCCESS S AH K S EH S
THREE TH R IY
TWO T UW

**speech_commands.lm -**

Language model created by QuickLM on Thu Apr 19 17:51:57 EDT 2018

The model is in standard ARPA format, designed by Doug Paul while he was at MITRE.

The code that was used to produce this language model is available in Open Source.
Please visit http://www.speech.cs.cmu.edu/tools/ for more information

The (fixed) discount mass is 0.5. The backoffs are computed using the ratio method.
This model based on a corpus of 28 sentences and 30 words

\data\
ngram 1=30
ngram 2=56
ngram 3=28

\1-grams:
-0.7782 </s> -0.3010
-0.7782 <s> -0.2218
-2.2253 ALPHA -0.2218
-2.2253 BISHOP -0.2218
-2.2253 BRAVO -0.2218
-2.2253 CHARLIE -0.2218
-2.2253 CONTINUE -0.2218
-2.2253 DELTA -0.2218
-2.2253 ECHO -0.2218
-2.2253 EIGHT -0.2218
-2.2253 FAILURE -0.2218
-2.2253 FIVE -0.2218
-2.2253 FOUR -0.2218
-2.2253 FOXTROT -0.2218
-2.2253 GOLF -0.2218
-2.2253 HOTEL -0.2218
-2.2253 KING -0.2218
-2.2253 KNIGHT -0.2218
-2.2253 ONE -0.2218
-2.2253 PAUSE -0.2218
-2.2253 PAWN -0.2218
-2.2253 QUEEN -0.2218
-2.2253 RESET -0.2218
-2.2253 ROOK -0.2218
-2.2253 SEVEN -0.2218
-2.2253 SHUTDOWN -0.2218
-2.2253 SIX -0.2218
-2.2253 SUCCESS -0.2218
-2.2253 THREE -0.2218
-2.2253 TWO -0.2218

\2-grams:
-1.7482 <s> ALPHA 0.0000
-1.7482 <s> BISHOP 0.0000
-1.7482 <s> BRAVO 0.0000
-1.7482 <s> CHARLIE 0.0000
-1.7482 <s> CONTINUE 0.0000
-1.7482 <s> DELTA 0.0000
-1.7482 <s> ECHO 0.0000
-1.7482 <s> EIGHT 0.0000
-1.7482 <s> FAILURE 0.0000
-1.7482 <s> FIVE 0.0000
-1.7482 <s> FOUR 0.0000
-1.7482 <s> FOXTROT 0.0000
-1.7482 <s> GOLF 0.0000
-1.7482 <s> HOTEL 0.0000
-1.7482 <s> KING 0.0000
-1.7482 <s> KNIGHT 0.0000
-1.7482 <s> ONE 0.0000
-1.7482 <s> PAUSE 0.0000
-1.7482 <s> PAWN 0.0000
-1.7482 <s> QUEEN 0.0000
-1.7482 <s> RESET 0.0000
-1.7482 <s> ROOK 0.0000
-1.7482 <s> SEVEN 0.0000
-1.7482 <s> SHUTDOWN 0.0000

-1.7482 <s> SIX 0.0000
-1.7482 <s> SUCCESS 0.0000
-1.7482 <s> THREE 0.0000
-1.7482 <s> TWO 0.0000
-0.3010 ALPHA </s> -0.3010
-0.3010 BISHOP </s> -0.3010
-0.3010 BRAVO </s> -0.3010
-0.3010 CHARLIE </s> -0.3010
-0.3010 CONTINUE </s> -0.3010
-0.3010 DELTA </s> -0.3010
-0.3010 ECHO </s> -0.3010
-0.3010 EIGHT </s> -0.3010
-0.3010 FAILURE </s> -0.3010
-0.3010 FIVE </s> -0.3010
-0.3010 FOUR </s> -0.3010
-0.3010 FOXTROT </s> -0.3010
-0.3010 GOLF </s> -0.3010
-0.3010 HOTEL </s> -0.3010
-0.3010 KING </s> -0.3010
-0.3010 KNIGHT </s> -0.3010
-0.3010 ONE </s> -0.3010
-0.3010 PAUSE </s> -0.3010
-0.3010 PAWN </s> -0.3010
-0.3010 QUEEN </s> -0.3010
-0.3010 RESET </s> -0.3010
-0.3010 ROOK </s> -0.3010
-0.3010 SEVEN </s> -0.3010
-0.3010 SHUTDOWN </s> -0.3010
-0.3010 SIX </s> -0.3010
-0.3010 SUCCESS </s> -0.3010
-0.3010 THREE </s> -0.3010
-0.3010 TWO </s> -0.3010

\3-grams:
-0.3010 <s> ALPHA </s>
-0.3010 <s> BISHOP </s>
-0.3010 <s> BRAVO </s>
-0.3010 <s> CHARLIE </s>
-0.3010 <s> CONTINUE </s>
-0.3010 <s> DELTA </s>
-0.3010 <s> ECHO </s>
-0.3010 <s> EIGHT </s>
-0.3010 <s> FAILURE </s>
-0.3010 <s> FIVE </s>
-0.3010 <s> FOUR </s>
-0.3010 <s> FOXTROT </s>
-0.3010 <s> GOLF </s>
-0.3010 <s> HOTEL </s>
-0.3010 <s> KING </s>
-0.3010 <s> KNIGHT </s>
-0.3010 <s> ONE </s>
-0.3010 <s> PAUSE </s>
-0.3010 <s> PAWN </s>
-0.3010 <s> QUEEN </s>
-0.3010 <s> RESET </s>
-0.3010 <s> ROOK </s>
-0.3010 <s> SEVEN </s>
-0.3010 <s> SHUTDOWN </s>
-0.3010 <s> SIX </s>
-0.3010 <s> SUCCESS </s>
-0.3010 <s> THREE </s>
-0.3010 <s> TWO </s>

\end\

**speech_commands.log_pronounce -**
ALPHA - Main
BISHOP - Main
BRAVO - Main
CHARLIE - Main

CONTINUE - Main
DELTA - Main
ECHO - Main
EIGHT - Main
FAILURE - Main
FIVE - Main
FOUR - Main
FOXTROT - Main
GOLF - Main
HOTEL - Main
KING - Main
KNIGHT - Main
ONE - Main
PAUSE - Main
PAWN - Main
QUEEN - Main
RESET - Main
ROOK - Main
SEVEN - Main
SHUTDOWN - Main
SIX - Main
SUCCESS - Main
THREE - Main
TWO - Main

**speech_commands.sent -**
<s> ONE </s>
<s> TWO </s>
<s> THREE </s>
<s> FOUR </s>
<s> FIVE </s>
<s> SIX </s>
<s> SEVEN </s>
<s> EIGHT </s>
<s> ALPHA </s>
<s> BRAVO </s>
<s> CHARLIE </s>
<s> DELTA </s>
<s> ECHO </s>
<s> FOXTROT </s>
<s> GOLF </s>
<s> HOTEL </s>
<s> PAWN </s>
<s> ROOK </s>
<s> KNIGHT </s>
<s> KING </s>
<s> QUEEN </s>
<s> BISHOP </s>
<s> PAUSE </s>
<s> CONTINUE </s>
<s> SHUTDOWN </s>
<s> RESET </s>
<s> SUCCESS </s>
<s> FAILURE </s>

**speech_commands.vocab -**
ALPHA
BISHOP
BRAVO
CHARLIE
CONTINUE
DELTA
ECHO
EIGHT
FAILURE
FIVE
FOUR
FOXTROT
GOLF

HOTEL
KING
KNIGHT
ONE
PAUSE
PAWN
QUEEN
RESET
ROOK
SEVEN
SHUTDOWN
SIX
SUCCESS
THREE
TWO

**Files Utilized, Modified, and Developed by Connor Desmond**

**Game.py**

```
from __future__ import print_function
from Board import Board
from Null import Null


class Game():
    def __init__(self):

        self.initial_board = Board()
        self.piece_dict = {}
        self.white_piece_dict = {}
        self.black_piece_dict = {}
        self.killed_piece = False

    def label_to_loc(self,label):
        loc = []
        letters = ['a','b','c','d','e','f','g','h']
        numbers = ['1','2','3','4','5','6','7','8']


        for i in range(8):
            if(letters[i]==label[0]):
                loc.append(i)
        for j in range(8):
            if(numbers[j]==label[1]):
                loc.append(j)
        return loc

    def print_board(self):

        for i in range(8):
            for j in range(8):
                print(self.initial_board.board[i][j].return_piece().identity(), end = " ")
            print("\n")
        for i in range(8):
            for j in range(8):
                print(self.initial_board.board[i][j].return_piece().return_color_string(), end = " ")
            print("\n")
        for i in range(8):
            for j in range(8):
                print(self.initial_board.board[i][j].return_piece().return_labelp(), end = " ")
            print("\n")
        for i in range(8):
            for j in range(8):
                print(self.initial_board.board[i][j].return_loc(), end = " ")
            print("\n")
        for i in range(8):
            for j in range(8):
                print(self.initial_board.board[i][j].return_color(), end = " ")
```

```
        print("\n")

    def create_piece_dict(self):
        self.create_white_piece_dict()
        self.create_black_piece_dict()
    def create_white_piece_dict(self):
        i = 0
        j = 0
        #Pawns
        for n in range(8):
            i,j = self.initial_board.xy_to_ij(n,1)
            key = self.initial_board.board[i][j].return_piece().return_labelp()
            value = self.initial_board.board[i][j].return_piece().return_locp()
            self.white_piece_dict[key] = value
        #Queen
        i,j = self.initial_board.xy_to_ij(3,0)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.white_piece_dict[key] = value
        #Rooks
        i,j = self.initial_board.xy_to_ij(0,0)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.white_piece_dict[key] = value
        i,j = self.initial_board.xy_to_ij(7,0)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.white_piece_dict[key] = value
        #Bishops
        i,j = self.initial_board.xy_to_ij(2,0)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.white_piece_dict[key] = value
        i,j = self.initial_board.xy_to_ij(5,0)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.white_piece_dict[key] = value
        #King
        i,j = self.initial_board.xy_to_ij(4,0)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.white_piece_dict[key] = value
        #Knights
        i,j = self.initial_board.xy_to_ij(1,0)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.white_piece_dict[key] = value
        i,j = self.initial_board.xy_to_ij(6,0)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.white_piece_dict[key] = value
    def create_black_piece_dict(self):
        i = 0
        j = 0
        #Pawns
        for n in range(8):
            i,j = self.initial_board.xy_to_ij(n,6)
            key = self.initial_board.board[i][j].return_piece().return_labelp()
            value = self.initial_board.board[i][j].return_piece().return_locp()
            self.black_piece_dict[key] = value
        #Queen
        i,j = self.initial_board.xy_to_ij(3,7)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.black_piece_dict[key] = value
        #Rooks
        i,j = self.initial_board.xy_to_ij(0,7)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.black_piece_dict[key] = value
```

```
        i,j = self.initial_board.xy_to_ij(7,7)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.black_piece_dict[key] = value
        #Bishops
        i,j = self.initial_board.xy_to_ij(2,7)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.black_piece_dict[key] = value
        i,j = self.initial_board.xy_to_ij(5,7)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.black_piece_dict[key] = value
        #King
        i,j = self.initial_board.xy_to_ij(4,7)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.black_piece_dict[key] = value
        #Knights
        i,j = self.initial_board.xy_to_ij(1,7)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.black_piece_dict[key] = value
        i,j = self.initial_board.xy_to_ij(6,7)
        key = self.initial_board.board[i][j].return_piece().return_labelp()
        value = self.initial_board.board[i][j].return_piece().return_locp()
        self.black_piece_dict[key] = value

    #function which prints the contents of white and black dictionaries
    def print_piece_dict(self):
        for key,value in self.white_piece_dict.items():
            print(key,":",value)
        print('\n')
        for key,value in self.black_piece_dict.items():
            print(key,":",value)

    #function which iterates through the dictionary keys and returns if there is
    #a match to label
    def check_piece_dict(self,label,white_turn):
        if(white_turn):
            for key in self.white_piece_dict.keys():
                if(label==key):
                    return True
        else:
            for key in self.black_piece_dict.keys():
                if(label==key):
                    return True
        return False
    #function which generates a list of valid moves for a piece which wants to move
    def generate_moves(self,locp,white_turn):
        i = 0
        j = 0
        move_list = []
        piece_type = " "
        i,j = self.initial_board.xy_to_ij(locp[0],locp[1])
        piece = self.initial_board.board[i][j].return_piece()
        piece_type = piece.identity()
        currentX = piece.return_locp()[0]
        currentY = piece.return_locp()[1]

        if(piece_type == "Pawn"):
            if(white_turn):
                d1 = self.pawn_d1(currentX,currentY)
                u = self.pawn_u(currentX,currentY)
                u2 = self.pawn_u2(currentX,currentY)
                d2 = self.pawn_d2(currentX,currentY)

                if(d1):
                    move_list.append([currentX-1,currentY+1])
```

```
            if(u):
                move_list.append([currentX,currentY+1])
            if(u2):
                move_list.append([currentX,currentY+2])
            if(d2):
                move_list.append([currentX+1,currentY+1])
        else:
            d4 = self.pawn_d4(currentX,currentY)
            l  = self.pawn_l(currentX,currentY)
            l2 = self.pawn_l2(currentX,currentY)
            d3 = self.pawn_d3(currentX,currentY)

            if(d4):
                move_list.append([currentX-1,currentY-1])
            if(l):
                move_list.append([currentX,currentY-1])
            if(l2):
                move_list.append([currentX,currentY-2])
            if(d3):
                move_list.append([currentX+1,currentY-1])
    elif(piece_type == "Queen"):
        move_list = self.f_moves(currentX,currentY,move_list)
        move_list = self.d1_moves(currentX,currentY,move_list)
        move_list = self.u_moves(currentX,currentY,move_list)
        move_list = self.d2_moves(currentX,currentY,move_list)
        move_list = self.r_moves(currentX,currentY,move_list)
        move_list = self.d3_moves(currentX,currentY,move_list)
        move_list = self.l_moves(currentX,currentY,move_list)
        move_list = self.d4_moves(currentX,currentY,move_list)
    elif(piece_type == "Rook"):
        move_list = self.f_moves(currentX,currentY,move_list)
        move_list = self.u_moves(currentX,currentY,move_list)
        move_list = self.r_moves(currentX,currentY,move_list)
        move_list = self.l_moves(currentX,currentY,move_list)
    elif(piece_type == "Bishop"):
        move_list = self.d1_moves(currentX,currentY,move_list)
        move_list = self.d2_moves(currentX,currentY,move_list)
        move_list = self.d3_moves(currentX,currentY,move_list)
        move_list = self.d4_moves(currentX,currentY,move_list)
    elif(piece_type == "King"):
        move_list = self.f_move(currentX,currentY,move_list)
        move_list = self.d1_move(currentX,currentY,move_list)
        move_list = self.u_move(currentX,currentY,move_list)
        move_list = self.d2_move(currentX,currentY,move_list)
        move_list = self.r_move(currentX,currentY,move_list)
        move_list = self.d3_move(currentX,currentY,move_list)
        move_list = self.l_move(currentX,currentY,move_list)
        move_list = self.d4_move(currentX,currentY,move_list)
    elif(piece_type == "Knight"):
        move_list = self.n1_move(currentX,currentY,move_list)
        move_list = self.n2_move(currentX,currentY,move_list)
        move_list = self.n3_move(currentX,currentY,move_list)
        move_list = self.n4_move(currentX,currentY,move_list)
        move_list = self.n5_move(currentX,currentY,move_list)
        move_list = self.n6_move(currentX,currentY,move_list)
        move_list = self.n7_move(currentX,currentY,move_list)
        move_list = self.n8_move(currentX,currentY,move_list)

    return move_list

''' PAWN CARDINAL DIRECTION BOOLEAN FUNCTIONS '''

def pawn_u(self,currentX,currentY):
    i = 0
    newi = 0
    j = 0
    newj = 0
    i,j = self.initial_board.xy_to_ij(currentX,currentY)
    if(currentY+1 > 7):
        return False
```

```
        else:
            newi,newj = self.initial_board.xy_to_ij(currentX,currentY+1)

        enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

        if(enemy_color == "None"):
            return True
        else:
            return False

    def pawn_l(self,currentX,currentY):
        i = 0
        newi = 0
        j = 0
        newj = 0
        i,j = self.initial_board.xy_to_ij(currentX,currentY)
        if(currentY-1 < 0):
            return False
        else:
            newi,newj = self.initial_board.xy_to_ij(currentX,currentY-1)

        enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

        if(enemy_color == "None"):
            return True
        else:
            return False
    def pawn_u2(self,currentX,currentY):
        i = 0
        newi = 0
        j = 0
        newj = 0
        i,j = self.initial_board.xy_to_ij(currentX,currentY)
        if(currentY+2 > 7):
            return False
        else:
            newi,newj = self.initial_board.xy_to_ij(currentX,currentY+2)

        enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

        if(self.initial_board.board[i][j].return_piece().init_loc == False):
            return False
        if(enemy_color == "None"):
            return True
        else:
            return False
    def pawn_l2(self,currentX,currentY):
        i = 0
        newi = 0
        j = 0
        newj = 0
        i,j = self.initial_board.xy_to_ij(currentX,currentY)
        if(currentY-2 < 0):
            return False
        else:
            newi,newj = self.initial_board.xy_to_ij(currentX,currentY-2)

        enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

        if(self.initial_board.board[i][j].return_piece().init_loc == False):
            return False
        if(enemy_color == "None"):
            return True
        else:
            return False

    def pawn_d1(self,currentX,currentY):
        i = 0
        newi = 0
        j = 0
```

```
        newj = 0
        i,j = self.initial_board.xy_to_ij(currentX,currentY)
        if(currentX-1 < 0 or currentY+1 > 7):
            return False
        else:
            newi,newj = self.initial_board.xy_to_ij(currentX-1,currentY+1)

        piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
        enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

        if(enemy_color == "None"):
            return False
        else:
            if(enemy_color == piece_color):
                return False
            else:
                return True
    def pawn_d2(self,currentX,currentY):
        i = 0
        newi = 0
        j = 0
        newj = 0
        i,j = self.initial_board.xy_to_ij(currentX,currentY)
        if(currentX+1 > 7 or currentY+1 > 7):
            return False
        else:
            newi,newj = self.initial_board.xy_to_ij(currentX+1,currentY+1)

        piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
        enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

        if(enemy_color == "None"):
            return False
        else:
            if(enemy_color == piece_color):
                return False
            else:
                return True
    def pawn_d3(self,currentX,currentY):
        i = 0
        newi = 0
        j = 0
        newj = 0
        i,j = self.initial_board.xy_to_ij(currentX,currentY)
        if(currentX+1 > 7 or currentY-1 < 0):
            return False
        else:
            newi,newj = self.initial_board.xy_to_ij(currentX+1,currentY-1)

        piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
        enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

        if(enemy_color == "None"):
            return False
        else:
            if(enemy_color == piece_color):
                return False
            else:
                return True
    def pawn_d4(self,currentX,currentY):
        i = 0
        newi = 0
        j = 0
        newj = 0
        i,j = self.initial_board.xy_to_ij(currentX,currentY)
        if(currentX-1 < 0 or currentY-1 < 0):
            return False
        else:
            newi,newj = self.initial_board.xy_to_ij(currentX-1,currentY-1)
```

```
    piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
    enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

    if(enemy_color == "None"):
        return False
    else:
        if(enemy_color == piece_color):
            return False
        else:
            return True
'"END OF PAWN CARDINAL DIRECTION BOOLEAN FUNCTIONS "'
'"START OF KNIGHT SPECIFIC MOVEMENT CHECKS"'
def n1_move(self,currentX,currentY,move_list):
    X = currentX-1
    Y = currentY+2
    i = 0
    j = 0
    i,j = self.initial_board.xy_to_ij(currentX,currentY)
    piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
    check = self.n_check(X,Y,piece_color)
    if(check):
        move_list.append([X,Y])
        return move_list
    else:
        return move_list
def n2_move(self,currentX,currentY,move_list):
    X = currentX+1
    Y = currentY+2
    i = 0
    j = 0
    i,j = self.initial_board.xy_to_ij(currentX,currentY)
    piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
    check = self.n_check(X,Y,piece_color)
    if(check):
        move_list.append([X,Y])
        return move_list
    else:
        return move_list
def n3_move(self,currentX,currentY,move_list):
    X = currentX+2
    Y = currentY+1
    i = 0
    j = 0
    i,j = self.initial_board.xy_to_ij(currentX,currentY)
    piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
    check = self.n_check(X,Y,piece_color)
    if(check):
        move_list.append([X,Y])
        return move_list
    else:
        return move_list
def n4_move(self,currentX,currentY,move_list):
    X = currentX+2
    Y = currentY-1
    i = 0
    j = 0
    i,j = self.initial_board.xy_to_ij(currentX,currentY)
    piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
    check = self.n_check(X,Y,piece_color)
    if(check):
        move_list.append([X,Y])
        return move_list
    else:
        return move_list
def n5_move(self,currentX,currentY,move_list):
    X = currentX+1
    Y = currentY-2
    i = 0
    j = 0
    i,j = self.initial_board.xy_to_ij(currentX,currentY)
```

```
        piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
        check = self.n_check(X,Y,piece_color)
        if(check):
            move_list.append([X,Y])
            return move_list
        else:
            return move_list
    def n6_move(self,currentX,currentY,move_list):
        X = currentX-1
        Y = currentY-2
        i = 0
        j = 0
        i,j = self.initial_board.xy_to_ij(currentX,currentY)
        piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
        check = self.n_check(X,Y,piece_color)
        if(check):
            move_list.append([X,Y])
            return move_list
        else:
            return move_list
    def n7_move(self,currentX,currentY,move_list):
        X = currentX-2
        Y = currentY-1
        i = 0
        j = 0
        i,j = self.initial_board.xy_to_ij(currentX,currentY)
        piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
        check = self.n_check(X,Y,piece_color)
        if(check):
            move_list.append([X,Y])
            return move_list
        else:
            return move_list
    def n8_move(self,currentX,currentY,move_list):
        X = currentX-2
        Y = currentY+1
        i = 0
        j = 0
        i,j = self.initial_board.xy_to_ij(currentX,currentY)
        piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
        check = self.n_check(X,Y,piece_color)
        if(check):
            move_list.append([X,Y])
            return move_list
        else:
            return move_list
'"END OF KNIGHT SPECIFIC MOVEMENT CHECKS"'
'"START OF KING SPECIFIC MOVEMENT CHECKS"'
    def f_move(self,currentX,currentY,move_list):
        X = currentX
        Y = currentY
        i = 0
        j = 0
        i,j = self.initial_board.xy_to_ij(X,Y)
        piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
        check,enemy_piece = self.f_check(X,Y,piece_color)
        if(check):
            move_list.append([X-1,Y])
            return move_list
        else:
            return move_list
    def d1_move(self,currentX,currentY,move_list):
        X = currentX
        Y = currentY
        i = 0
        j = 0
        i,j = self.initial_board.xy_to_ij(X,Y)
        piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
        check,enemy_piece = self.d1_check(X,Y,piece_color)
        if(check):
```

```
            move_list.append([X-1,Y+1])
            return move_list
        else:
            return move_list
    def u_move(self,currentX,currentY,move_list):
        X = currentX
        Y = currentY
        i = 0
        j = 0
        i,j = self.initial_board.xy_to_ij(X,Y)
        piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
        check,enemy_piece = self.u_check(X,Y,piece_color)
        if(check):
            move_list.append([X,Y+1])
            return move_list
        else:
            return move_list

    def d2_move(self,currentX,currentY,move_list):
        X = currentX
        Y = currentY
        i = 0
        j = 0
        i,j = self.initial_board.xy_to_ij(X,Y)
        piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
        check,enemy_piece = self.d2_check(X,Y,piece_color)
        if(check):
            move_list.append([X+1,Y+1])
            return move_list
        else:
            return move_list
    def r_move(self,currentX,currentY,move_list):
        X = currentX
        Y = currentY
        i = 0
        j = 0
        i,j = self.initial_board.xy_to_ij(X,Y)
        piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
        check,enemy_piece = self.r_check(X,Y,piece_color)
        if(check):
            move_list.append([X+1,Y])
            return move_list
        else:
            return move_list
    def d3_move(self,currentX,currentY,move_list):
        X = currentX
        Y = currentY
        i = 0
        j = 0
        i,j = self.initial_board.xy_to_ij(X,Y)
        piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
        check,enemy_piece = self.d3_check(X,Y,piece_color)
        if(check):
            move_list.append([X+1,Y-1])
            return move_list
        else:
            return move_list
    def l_move(self,currentX,currentY,move_list):
        X = currentX
        Y = currentY
        i = 0
        j = 0
        i,j = self.initial_board.xy_to_ij(X,Y)
        piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
        check,enemy_piece = self.l_check(X,Y,piece_color)
        if(check):
            move_list.append([X,Y-1])
            return move_list
        else:
            return move_list
```

```
def d4_move(self,currentX,currentY,move_list):
    X = currentX
    Y = currentY
    i = 0
    j = 0
    i,j = self.initial_board.xy_to_ij(X,Y)
    piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
    check,enemy_piece = self.d4_check(X,Y,piece_color)
    if(check):
        move_list.append([X-1,Y-1])
        return move_list
    else:
        return move_list
 '"END OF KING SPECIFIC MOVEMENT CHECKS"'
 '"START OF CARDINAL DIRECTION CHECKS"'
def d4_check(self,currentX,currentY,piece_color):
    i = 0
    newi = 0
    j = 0
    newj = 0
    i,j = self.initial_board.xy_to_ij(currentX,currentY)

    if(currentX-1 < 0 or currentY-1 < 0):
        return False,False
    else:
        newi,newj = self.initial_board.xy_to_ij(currentX-1,currentY-1)

    enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

    if(enemy_color == "None"):
        return True,False
    else:
        if(enemy_color == piece_color):
            return False,False
        else:
            return True,True
def d3_check(self,currentX,currentY,piece_color):
    i = 0
    newi = 0
    j = 0
    newj = 0
    i,j = self.initial_board.xy_to_ij(currentX,currentY)

    if(currentX+1 > 7 or currentY-1 < 0):
        return False,False
    else:
        newi,newj = self.initial_board.xy_to_ij(currentX+1,currentY-1)

    enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

    if(enemy_color == "None"):
        return True,False
    else:
        if(enemy_color == piece_color):
            return False,False
        else:
            return True,True
def d2_check(self,currentX,currentY,piece_color):
    i = 0
    newi = 0
    j = 0
    newj = 0
    i,j = self.initial_board.xy_to_ij(currentX,currentY)

    if(currentX+1 > 7 or currentY+1 > 7):
        return False,False
    else:
        newi,newj = self.initial_board.xy_to_ij(currentX+1,currentY+1)

    enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()
```

```
    if(enemy_color == "None"):
        return True,False
    else:
        if(enemy_color == piece_color):
            return False,False
        else:
            return True,True
def d1_check(self,currentX,currentY,piece_color):
    i = 0
    newi = 0
    j = 0
    newj = 0
    i,j = self.initial_board.xy_to_ij(currentX,currentY)

    if(currentX-1 < 0 or currentY+1 > 7):
        return False,False
    else:
        newi,newj = self.initial_board.xy_to_ij(currentX-1,currentY+1)

    enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

    if(enemy_color == "None"):
        return True,False
    else:
        if(enemy_color == piece_color):
            return False,False
        else:
            return True,True
def l_check(self,currentX,currentY,piece_color):
    i = 0
    newi = 0
    j = 0
    newj = 0
    i,j = self.initial_board.xy_to_ij(currentX,currentY)

    if(currentY-1 < 0):
        return False,False
    else:
        newi,newj = self.initial_board.xy_to_ij(currentX,currentY-1)

    enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

    if(enemy_color == "None"):
        return True,False
    else:
        if(enemy_color == piece_color):
            return False,False
        else:
            return True,True
def u_check(self,currentX,currentY,piece_color):
    i = 0
    newi = 0
    j = 0
    newj = 0
    i,j = self.initial_board.xy_to_ij(currentX,currentY)

    if(currentY+1 > 7):
        return False,False
    else:
        newi,newj = self.initial_board.xy_to_ij(currentX,currentY+1)

    enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

    if(enemy_color == "None"):
        return True,False
    else:
        if(enemy_color == piece_color):
            return False,False
        else:
```

```
            return True,True
    def r_check(self,currentX,currentY,piece_color):
        i = 0
        newi = 0
        j = 0
        newj = 0
        i,j = self.initial_board.xy_to_ij(currentX,currentY)

        if(currentX+1 > 7):
            return False,False
        else:
            newi,newj = self.initial_board.xy_to_ij(currentX+1,currentY)

        enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()


        if(enemy_color == "None"):
            return True,False
        else:
            if(enemy_color == piece_color):
                return False,False
            else:
                return True,True
    def f_check(self,currentX,currentY,piece_color):
        i = 0
        newi = 0
        j = 0
        newj = 0
        i,j = self.initial_board.xy_to_ij(currentX,currentY)

        if(currentX-1 < 0):
            return False,False
        else:
            newi,newj = self.initial_board.xy_to_ij(currentX-1,currentY)

        enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

        if(enemy_color == "None"):
            return True,False
        else:
            if(enemy_color == piece_color):
                return False,False
            else:
                return True,True
     '"END of CARDINAL DIRECTION CHECKS"'
    #function which does a check for all knight movements
    def n_check(self,X,Y,piece_color):
        i = 0
        newi = 0
        j = 0
        newj = 0
        i,j = self.initial_board.xy_to_ij(X,Y)
        if(X < 0 or X > 7 or Y < 0 or Y > 7):
            return False
        else:
            newi,newj = self.initial_board.xy_to_ij(X,Y)

        enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()

        if(enemy_color == "None"):
            return True
        else:
            if(enemy_color == piece_color):
                return False
            else:
                return True
    #START OF QUEEN/BISHOP/ROOK MOVEMENT CHECKS
    def d4_moves(self,currentX,currentY,move_list):
        X = currentX
        Y = currentY
```

```
    i = 0
    j = 0
    i,j = self.initial_board.xy_to_ij(X,Y)
    piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
    check,enemy_piece = self.d4_check(X,Y,piece_color)
    while(check):
        move_list.append([X-1,Y-1])
        Y = Y - 1
        X = X - 1

        if(enemy_piece):
            break
        check,enemy_piece = self.d4_check(X,Y,piece_color)
    return move_list
def d3_moves(self,currentX,currentY,move_list):
    X = currentX
    Y = currentY
    i = 0
    j = 0
    i,j = self.initial_board.xy_to_ij(X,Y)
    piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
    check,enemy_piece = self.d3_check(X,Y,piece_color)
    while(check):
        move_list.append([X+1,Y-1])
        Y = Y - 1
        X = X + 1

        if(enemy_piece):
            break
        check,enemy_piece = self.d3_check(X,Y,piece_color)
    return move_list
def d2_moves(self,currentX,currentY,move_list):
    X = currentX
    Y = currentY
    i = 0
    j = 0
    i,j = self.initial_board.xy_to_ij(X,Y)
    piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
    check,enemy_piece = self.d2_check(X,Y,piece_color)
    while(check):
        move_list.append([X+1,Y+1])
        Y = Y + 1
        X = X + 1

        if(enemy_piece):
            break
        check,enemy_piece = self.d2_check(X,Y,piece_color)
    return move_list
def d1_moves(self,currentX,currentY,move_list):
    X = currentX
    Y = currentY
    i = 0
    j = 0
    i,j = self.initial_board.xy_to_ij(X,Y)
    piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
    check,enemy_piece = self.d1_check(X,Y,piece_color)
    while(check):

        move_list.append([X-1,Y+1])
        Y = Y + 1
        X = X - 1

        if(enemy_piece):
            break
        check,enemy_piece = self.d1_check(X,Y,piece_color)
    return move_list
def l_moves(self,currentX,currentY,move_list):
    X = currentX
    Y = currentY
    i = 0
```

```
    j = 0
    i,j = self.initial_board.xy_to_ij(X,Y)
    piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
    check,enemy_piece = self.l_check(X,Y,piece_color)
    while(check):
        move_list.append([X,Y-1])
        Y = Y - 1
        if(enemy_piece):
            break
        check,enemy_piece = self.l_check(X,Y,piece_color)
    return move_list
def u_moves(self,currentX,currentY,move_list):
    X = currentX
    Y = currentY
    i = 0
    j = 0
    i,j = self.initial_board.xy_to_ij(X,Y)
    piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
    check,enemy_piece = self.u_check(X,Y,piece_color)
    while(check):
        move_list.append([X,Y+1])
        Y = Y + 1
        if(enemy_piece):
            break
        check,enemy_piece = self.u_check(X,Y,piece_color)
    return move_list
def f_moves(self,currentX,currentY,move_list):
    X = currentX
    Y = currentY
    i = 0
    j = 0
    i,j = self.initial_board.xy_to_ij(X,Y)
    piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
    check,enemy_piece = self.f_check(X,Y,piece_color)
    while(check):
        move_list.append([X-1,Y])
        X = X - 1
        if(enemy_piece):
            break
        check,enemy_piece = self.f_check(X,Y,piece_color)
    return move_list

def r_moves(self,currentX,currentY,move_list):
    X = currentX
    Y = currentY
    i = 0
    j = 0
    i,j = self.initial_board.xy_to_ij(X,Y)
    piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
    check,enemy_piece = self.r_check(X,Y,piece_color)
    while(check):
        move_list.append([X+1,Y])
        X = X + 1
        if(enemy_piece):
            break
        check,enemy_piece = self.r_check(X,Y,piece_color)
    return move_list
'"END OF KING/BISHOP/ROOK MOVEMENT CHECKS"'
#function that was used for testing purposes
def place_piece(self,piece,loc,colorp):
    i = 0
    j = 0
    i,j = self.initial_board.xy_to_ij(loc[0],loc[1])
    self.initial_board.board[i][j].change_square_state()
    self.initial_board.board[i][j].set_piece(piece)
    self.initial_board.board[i][j].return_piece().set_locp(loc[0],loc[1])
    self.initial_board.board[i][j].return_piece().set_color_string(colorp)
    self.initial_board.board[i][j].return_piece().set_labelp("PT")
#function which is used to remove a key value pair from a either the white or black dictionary.
def remove_from_dict(self,label,white_turn):
```

```
            if(white_turn):
                        del self.black_piece_dict[label]
            else:
                        del self.white_piece_dict[label]
#function which determines if the king piece has been killed or not
def check_king_state(self,white_turn):
            if(white_turn):
                for key in self.black_piece_dict.keys():
            if('K'==key):
                            return True
            else:
                for key in self.white_piece_dict.keys():
            if('K'==key):
                            return True
            return False
#function which returns true if a move is valid false otherwise
def check_move(self,label,loc,white_turn):
             self.killed_piece = False
    if(self.check_piece_dict(label,white_turn)):
        if(white_turn):
            current_loc = self.white_piece_dict[label]
        else:
            current_loc = self.black_piece_dict[label]
        i,j,newi,newj = 0,0,0,0
                i,j = self.initial_board.xy_to_ij(current_loc[0],current_loc[1])
                newi,newj = self.initial_board.xy_to_ij(loc[0],loc[1])
                piece_color = self.initial_board.board[i][j].return_piece().return_color_string()
                enemy_color = self.initial_board.board[newi][newj].return_piece().return_color_string()
        for move in self.generate_moves(current_loc,white_turn):
            if(move == loc):
                if(enemy_color == "None"):
                            self.killed_piece = False

                            else:
                                        if(piece_color == enemy_color):
                                            self.killed_piece = False

                                        else:
                                            self.killed_piece = True

                            return True,self.killed_piece
        return False,self.killed_piece
    else:
        return False,self.killed_piece
#function which takes care of moving a piece and updating the board state
def move_piece(self,label,loc_label,white_turn):
    i = 0
    j = 0
    newi = 0
    newj = 0
    current_init_loc = True
    loc = self.label_to_loc(loc_label)
             if(self.check_piece_dict(label,white_turn)):
                if(white_turn):
            current_loc = self.white_piece_dict[label]
                else:
            current_loc = self.black_piece_dict[label]
             else:
                print("Invalid Move")
        return False
    i,j = self.initial_board.xy_to_ij(current_loc[0],current_loc[1])
    newi,newj = self.initial_board.xy_to_ij(loc[0],loc[1])
    piece_type = self.initial_board.board[i][j].return_piece().identity()
    check,isKilled = self.check_move(label,loc,white_turn)
             victim_label = self.initial_board.board[newi][newj].return_piece().return_labelp()
             print(isKilled)

    if isKilled:
        self.remove_from_dict(victim_label,white_turn)
```

```
        if(check):
            if( piece_type == "Pawn"):
                self.initial_board.board[i][j].return_piece().set_init_loc(False)
                current_init_loc = self.initial_board.board[i][j].return_piece().init_loc
                  #if(abs(loc[1]-current_loc[1])==2):#just for pawns

            current_piece = self.initial_board.board[i][j].return_piece()
            current_piece_color = self.initial_board.board[i][j].return_piece().return_colorp()
                    current_color_string = self.initial_board.board[i][j].return_piece().return_color_string()
            current_labelp = self.initial_board.board[i][j].return_piece().return_labelp()


            self.initial_board.board[i][j].change_square_state()
            self.initial_board.board[newi][newj].change_square_state()

            self.initial_board.board[i][j].set_piece(Null())
            self.initial_board.board[i][j].return_piece().set_labelp("Null")
            self.initial_board.board[i][j].return_piece().set_locp(current_loc[0],current_loc[1])
            self.initial_board.board[i][j].return_piece().set_color_string("None")

            self.initial_board.board[newi][newj].set_piece(current_piece)
            self.initial_board.board[newi][newj].return_piece().set_colorp(current_piece_color)
            self.initial_board.board[newi][newj].return_piece().set_labelp(current_labelp)
            self.initial_board.board[newi][newj].return_piece().set_locp(loc[0],loc[1])
            self.initial_board.board[newi][newj].return_piece().set_color_string(current_color_string)
            if (piece_type == "Pawn"):
                self.initial_board.board[newi][newj].return_piece().set_init_loc(current_init_loc)

            if(white_turn):
                self.white_piece_dict[label] = loc
            else:
                self.black_piece_dict[label] = loc
            return True
        else:
            print("Invalid Move")
            return False
```

**Board.py**

```
from Pawn import Pawn
from Square import Square
from Null import Null
from Queen import Queen
from Rook import Rook
from Bishop import Bishop
from King import King
from Knight import Knight

class Board():

    def __init__(self):
        letters = ['a','b','c','d','e','f','g','h']
        numbers = ['1','2','3','4','5','6','7','8']

        self.board = []

        for j in range(8):
            column = []
            for i in range(8):
                column.append(0)
            self.board.append(column)
        x = 0
        y = 0

        for i in range(8):

            for j in range(8):
                self.board[i][j] = Square()
                x,y = self.ij_to_xy(i,j)

                self.board[i][j].set_piece(Null())
```

```
            self.board[i][j].return_piece().change_colorp()
            self.board[i][j].set_loc(x,y)
            self.board[i][j].set_label(letters[x]+numbers[y])
            self.board[i][j].return_piece().set_labelp("Null")
            self.board[i][j].return_piece().set_locp(x,y)


            if(i%2==0):
                if(j%2==0):
                    self.board[i][j].change_color()
            else:
                if(j%2!=0):
                    self.board[i][j].change_color()

    def initialize_board(self):
        i = 0
        j = 0
        pawn_labels = ["P1","P2","P3","P4","P5","P6","P7","P8"]
        queen_label = "Q"
        rook_labels = ["R1", "R2"]
        bishop_labels = ["B1","B2"]
        king_label = "K"
        knight_labels = ["N1","N2"]

        #White Pawns
        for n in range(8):
            i,j = self.xy_to_ij(n,1)
            self.board[i][j].change_square_state()
            self.board[i][j].set_piece(Pawn())
            self.board[i][j].return_piece().set_labelp(pawn_labels[n])
            self.board[i][j].return_piece().set_locp(n,1)
            self.board[i][j].return_piece().set_color_string("White")

        #White Queen

        i,j = self.xy_to_ij(3,0)
        self.board[i][j].change_square_state()
        self.board[i][j].set_piece(Queen())
        self.board[i][j].return_piece().set_labelp(queen_label)
        self.board[i][j].return_piece().set_locp(3,0)
        self.board[i][j].return_piece().set_color_string("White")

        #White Rooks

        i,j = self.xy_to_ij(0,0)
        self.board[i][j].change_square_state()
        self.board[i][j].set_piece(Rook())
        self.board[i][j].return_piece().set_labelp(rook_labels[0])
        self.board[i][j].return_piece().set_locp(0,0)
        self.board[i][j].return_piece().set_color_string("White")

        i,j = self.xy_to_ij(7,0)
        self.board[i][j].change_square_state()
        self.board[i][j].set_piece(Rook())
        self.board[i][j].return_piece().set_labelp(rook_labels[1])
        self.board[i][j].return_piece().set_locp(7,0)
        self.board[i][j].return_piece().set_color_string("White")

        #White Bishops

        i,j = self.xy_to_ij(2,0)
        self.board[i][j].change_square_state()
        self.board[i][j].set_piece(Bishop())
        self.board[i][j].return_piece().set_labelp(bishop_labels[0])
        self.board[i][j].return_piece().set_locp(2,0)
        self.board[i][j].return_piece().set_color_string("White")

        i,j = self.xy_to_ij(5,0)
        self.board[i][j].change_square_state()
        self.board[i][j].set_piece(Bishop())
```

```
self.board[i][j].return_piece().set_labelp(bishop_labels[1])
self.board[i][j].return_piece().set_locp(5,0)
self.board[i][j].return_piece().set_color_string("White")

#White King

i,j = self.xy_to_ij(4,0)
self.board[i][j].change_square_state()
self.board[i][j].set_piece(King())
self.board[i][j].return_piece().set_labelp(king_label)
self.board[i][j].return_piece().set_locp(4,0)
self.board[i][j].return_piece().set_color_string("White")

#White Knights
i,j = self.xy_to_ij(1,0)
self.board[i][j].change_square_state()
self.board[i][j].set_piece(Knight())
self.board[i][j].return_piece().set_labelp(knight_labels[0])
self.board[i][j].return_piece().set_locp(1,0)
self.board[i][j].return_piece().set_color_string("White")

i,j = self.xy_to_ij(6,0)
self.board[i][j].change_square_state()
self.board[i][j].set_piece(Knight())
self.board[i][j].return_piece().set_labelp(knight_labels[1])
self.board[i][j].return_piece().set_locp(6,0)
self.board[i][j].return_piece().set_color_string("White")

#Black Pawns

for n in range(8):
    i,j = self.xy_to_ij(7-n,6)
    self.board[i][j].change_square_state()
    self.board[i][j].set_piece(Pawn())
    self.board[i][j].return_piece().change_colorp()
    self.board[i][j].return_piece().set_labelp(pawn_labels[n])
    self.board[i][j].return_piece().set_locp(7-n,6)
    self.board[i][j].return_piece().set_color_string("Black")

#Black Queen

i,j = self.xy_to_ij(3,7)
self.board[i][j].change_square_state()
self.board[i][j].set_piece(Queen())
self.board[i][j].return_piece().set_labelp(queen_label)
self.board[i][j].return_piece().set_locp(3,7)
self.board[i][j].return_piece().set_color_string("Black")

#Black Rooks

i,j = self.xy_to_ij(0,7)
self.board[i][j].change_square_state()
self.board[i][j].set_piece(Rook())
self.board[i][j].return_piece().set_labelp(rook_labels[1])
self.board[i][j].return_piece().set_locp(0,7)
self.board[i][j].return_piece().set_color_string("Black")

i,j = self.xy_to_ij(7,7)
self.board[i][j].change_square_state()
self.board[i][j].set_piece(Rook())
self.board[i][j].return_piece().set_labelp(rook_labels[0])
self.board[i][j].return_piece().set_locp(7,7)
self.board[i][j].return_piece().set_color_string("Black")

#Black Bishops

i,j = self.xy_to_ij(2,7)
self.board[i][j].change_square_state()
self.board[i][j].set_piece(Bishop())
self.board[i][j].return_piece().set_labelp(bishop_labels[1])
```

```
        self.board[i][j].return_piece().set_locp(2,7)
        self.board[i][j].return_piece().set_color_string("Black")

        i,j = self.xy_to_ij(5,7)
        self.board[i][j].change_square_state()
        self.board[i][j].set_piece(Bishop())
        self.board[i][j].return_piece().set_labelp(bishop_labels[0])
        self.board[i][j].return_piece().set_locp(5,7)
        self.board[i][j].return_piece().set_color_string("Black")


        #White King

        i,j = self.xy_to_ij(4,7)
        self.board[i][j].change_square_state()
        self.board[i][j].set_piece(King())
        self.board[i][j].return_piece().set_labelp(king_label)
        self.board[i][j].return_piece().set_locp(4,7)
        self.board[i][j].return_piece().set_color_string("Black")

        #White Knights
        i,j = self.xy_to_ij(1,7)
        self.board[i][j].change_square_state()
        self.board[i][j].set_piece(Knight())
        self.board[i][j].return_piece().set_labelp(knight_labels[1])
        self.board[i][j].return_piece().set_locp(1,7)
        self.board[i][j].return_piece().set_color_string("Black")

        i,j = self.xy_to_ij(6,7)
        self.board[i][j].change_square_state()
        self.board[i][j].set_piece(Knight())
        self.board[i][j].return_piece().set_labelp(knight_labels[0])
        self.board[i][j].return_piece().set_locp(6,7)
        self.board[i][j].return_piece().set_color_string("Black")


    def ij_to_xy(self,i,j):
        x = j
        y = 7 - i
        return x,y
    def xy_to_ij(self,x,y):
        i = 7 - y
        j = x
        return i,j
```

**Square.py**

```
from Piece import Piece
class Square():
    def __init__(self):
        self.hasPiece = False
        self.isBlack = True
        self.label = "a1"
        self.piece = Piece()

        self.locX = 0
        self.locY = 0
        self.loc = [self.locX,self.locY]

    def change_square_state(self):
        self.hasPiece = not self.hasPiece
    def return_loc(self):
        return self.loc

    def return_locX(self):
        return self.real_locX

    def return_locY(self):
        return self.locY

    def set_loc(self,X,Y):
```

```
        self.loc = [X,Y]
        self.locX = X
        self.locY = Y

    def set_locX(self,X):
        self.locX = X
        self.loc = [self.locX,self.locY]

    def set_locY(self,Y):
        self.locY = Y
        self.loc = [self.locX,self.locY]

    def change_color(self):
        self.isBlack = not self.isBlack
    def return_color(self):
        return self.isBlack
    def set_label(self,label):
        self.label = label
    def return_label(self):
        return self.label
    def return_square_state(self):
        return self.hasPiece
    def set_piece(self,piece):
        self.piece = piece
    def return_piece(self):
        return self.piece
```

**Piece.py**

```
class Piece():
    def __init__(self):
        self.labelp = " "
        self.isWhite = True
        self.color_string = "None"
        self.locXp = 0
        self.locYp = 0
        self.locp = [self.locXp,self.locYp]

    def identity(self):
        return self.__class__.__name__
    def set_labelp(self,label):
        self.labelp = label
    def return_labelp(self):
        return self.labelp

    def return_locp(self):
        return self.locp
    def set_locp(self,Xp,Yp):
        self.locp = [Xp,Yp]
        self.locX = Xp
        self.locY = Yp
    def return_color_string(self):
        return self.color_string
    def set_color_string(self,color):
        self.color_string = color

    def return_colorp(self):
        return self.isWhite
    def change_colorp(self):
        self.isWhite = not self.isWhite
    def set_colorp(self,colorp):
        self.isWhite = colorp
    def return_locXp(self):
        return self.locXp
    def return_locYp(self):
        return self.locYp
    def set_locXp(self,Xp):
        self.locXp = Xp
        self.locp = [self.locXp,self.locYp]
    def set_locYp(self,Yp):
```

```
        self.locYp = Yp
        self.locp = [self.locXp,self.locYp]
```

**Queen.py**

```
from Piece import Piece
class Queen(Piece):
    def __init__(self):
        Piece.__init__(self)
```

**Pawn.py**

```
from Piece import Piece
class Pawn(Piece):
    def __init__(self):
        Piece.__init__(self)
        self.init_loc = True

    def set_init_loc(self,init_loc):
        self.init_loc = init_loc
```

**Rook.py**

```
from Piece import Piece
class Rook(Piece):
    def __init__(self):
        Piece.__init__(self)
```

**Bishop.py**

```
from Piece import Piece
class Bishop(Piece):
    def __init__(self):
        Piece.__init__(self)
```

**Knight.py**

```
from Piece import Piece
class Knight(Piece):
    def __init__(self):
        Piece.__init__(self)
```

**King.py**

```
from Piece import Piece
class King(Piece):
    def __init__(self):
        Piece.__init__(self)
```

**Null,py**

```
from Piece import Piece
class Null(Piece):
    def __init__(self):
        Piece.__init__(self)
    #def print_n():
      #print("n")
```