Baxter Image Recognition Class Summary

**Setup for Block Recognition Project**

Follow the steps from the Rethink Robotics wiki to setup your computer for use with Baxter:

Download the files from the github repository: https://github.com/Kevin-Roberts/baxter\_image\_recognition/tree/master

Move all files from the “src” folder to a catkin workspace. For example ~/ros\_ws/src/baxter\_tools/scripts/.

Place blocks on the edge of the table near the desired box. PICTURE (maybe)

Run “./baxter.sh” to initialize the connection.

Run “rosrun baxter\_tools enable\_robot.py –e” to enable the robot.

Places a block of each color on the table in front of the box location for that colored block. Baxter will detect the box locations by finding the block of that color, then setting the box location to 0.1 meters behind that block.

Run “rosrun baxter\_tools master\_controller.py config” to setup the configuration file. This will calculate the table height and box locations. Then these values will be saved to config.txt. This file is human readable and can be created / edited manually as well.

Run “rosrun baxter\_tools master\_controller.py” to begin detecting and sorting blocks.

Report any problems or questions to Kevin-Roberts (my github ID) on github or by emailing kevinroberts\_5@hotmail.com

**Master Controller Class**

**Attributes**

right\_camera : image\_receiver() class initialized for the right arm’s camera. Enabled

left\_camera : image\_receiver() class initialized for the left arm’s camera. Disabled

head\_camera : image\_receiver() class initialized for the head camera. Disabled

move : move\_controller() class initialized for the right arm.

image\_processor : image\_processor() class used to process images.

block\_list : A dictionary where each key is a block color and the value is a list of Block objects of blocks of that color. For example the purple block list is block\_list[‘PURPLE’].

box\_pose : A dictionary where each color is a key and the value is the Pose used for the box of that color

**Methods**

\_\_init\_\_(self, setconfig=False)

* (optional) setconfig : Boolean, set to true for configuration mode (configure box locations and table height)
* Initializes all attributes and sets up the robot so that the right arm gripper / camera is used to sort blocks.

update\_home\_pose(self, pose)

* pose: New home position that is inputted into the move controller and image processor.

get\_home\_pose(self)

* Takes an image from the home position and inputs it into the image processor class

find\_blocks(self)

* Calls get\_home\_pose(), then adds a list of block objects of each color in block\_list

are\_blocks\_near(self, block)

* block: The block object that is being targeted
* This method checks if any other blocks are nearby the targeted block. Could be used to avoid block clusters. Currently unused.

get\_blocks(self, trials=2, miss\_per\_trial=2)

* (optional)trials : number of pictures to be taken from the home position before failure
* (optional)miss\_per\_trial : number of blocks missed before a new trial is started (by taking a new picture from the same home location)
* This is the main function for finding blocks. Set the home position using update\_home\_pose() then call this function to find blocks from the current home position.

Main method in file accessed with no arguments i.e. “rosrun baxter\_tools master\_controller.py”

* A while loop that switches between 6 different home positions and calls get\_blocks() from each position. This while loop runs continuously until killed by the user.

Main method in file accessed with arguments i.e. “rosrun baxter\_tools master\_controller.py -c”

* Sets up the config.txt file for the current environment. Finds box locations and configures the table height.

Commented out in Main method

* Test that can be used to get block classification color range accuracy.

**Move Controller Class**

**Attributes**

home\_pose : Pose object of the current home position

ns : String for the Kinematics solver subscriber topic

iksvc : rospy.ServiceProxy(self.ns, SolvePositionIK) object for the kinematics solver

ikreq : SolvePositionIKRequest() class.

hdr : header used for the PoseStamped object

arm : baxter\_interface limb controller initialized for the specified arm.

gripper : baxter\_interface gripper controller initialized for the specified arm.

table\_height : height of the table, this is set by the configuration infrared callback.

infrared\_topic : topic for the infrared subscriber.

infrared\_sub : variable to hold the infrared subscriber.

**Methods**

\_\_init\_\_(self, arm)

* arm : “right” or “left” used to specify the arm being initialized.
* This function initializes all of the attributes of the object for the desired arm, sets the movement speed to the maximum, and calibrates the gripper.

move\_to\_pose(self, poselist, move=True)

* Poselist : a list of Pose objects to be inputted into the IKSolver.
* (optional)move: Boolean that indicates if you actually want to move to that point. This can be set to false to check if there exists a valid joint position without actually moving to it.
* This is the main movement function that takes a list of Pose objects and solves for the joint position of each Pose and moves to each joint position in order. This function is called by several other move\_controller functions.
* If Tien can make Baxter always drop in a straight line, this is probably where he will do it.

move\_to\_home(self)

* Moves to the Pose specified by the self.home\_pose variable

raise\_up(self, pose)

* Raises Baxter’s arm up from the specified pose. Useful so that he doesn’t knock into other objects on his path to the next object or drop box.

pick\_at\_pose(self, pose)

* Moves to the specified Pose, pose.
* First goes to the position at home\_pose height and above the specified pose, then to a point 0.03 meters above the pose. Then to the pose, at this point the function closes the gripper to pick up the object then sleeps for 0.3 seconds to ensure the gripper is fully closed before moving onto the next step.
* THE FIRST POSITION, AT home\_pose HEIGHT could probably be removed (remove t1 from the poselist variable).

drop\_at\_pose(self, pose)

* Move to the specified pose and open the gripper to drop the object.

Update\_table\_height(self) (getInfrared and \_ir\_callback(self,data)

* Function called to determine the table\_height. This is done by calling getInfared(self) which sets up an Infrared sensor subscriber (\_ir\_callback() is the callback function for this subscriber) and drops Baxter’s Arm until the Infrared sensor is in range and then records the value to the self.table\_height variable.

new\_home\_pose(self)

* Function used to quickly create a Pose object, not crucial to operation.

**Image Receiver Class**

**Attributes**

camera\_name – String for camera name, ie right\_hand\_camera

camera\_controller – Baxter\_interface CameraController class object

image\_topic – Topic new images are published to by Baxter

image\_sub – The Subscriber class used when a new image is received

bridge – Used to convert Baxter’s image into OpenCV image

intrinsics\_topic/intrinsics\_sub – Topic and subscriber used to get the preset camera intrinsic for camera calibration

cv\_image – Most recent image received after being converted into a OpenCV image

raw\_image – Most recent raw image received from Baxter

**Methods**

\_img\_callback(self, data)

* The callback used to receive the image. Writes to self.raw\_image and self.cv\_image

\_intrinsics\_callback(self, data)

* The callback used to receive and format the camera preset camera intrinsics. Used for camera calibration

getIntrinsics(self)

* Sets up the camera intrinsics subscriber and returns the distortion coefficients and calibration matrix

getImage(self)

* Sets up the image subscriber for the desired camera, receives new image, then unsubscribes.

enabeCamera(self)

* Opens the camera, only 2 of the 3 cameras can be open at one time

disableCamera(self)

* Closes the camera.

Main()

* A test, not really needed anymore. Currently it sets up the image receiver, and receives an image from the right hand camera. Then it tries to find the PURPLE blocks in the image.

**Image Processor Class**

**Attributes**

blackout – variable used to specify the part of the image to black out so that no blocks are detected

home\_pose – variable of the default home\_pose position.

table\_height – The z coordinate of the table height

image\_topic – The topic used to display images on baxter’s xdisplay

bridge – used to conver from raw to OpenCV images or from OpenCV to raw images

pixels\_per\_meter – Scaled DEFAULT\_PIXELS\_PER\_METER value based on the table height and DEFAULT\_HOME\_HEIGHT

camera\_matrix – The Camera Matrix used for camera calibration

distortion – The distortion matrix used for camera calibration.

**Methods**

Update\_home\_pose(self,pose)

* This function should be used to update the home\_pose variable as opposed to writing self.home\_pose
* Updates the home\_pose variable and the pixels\_per\_meter value.

undistortImage(self, cv\_image)

* Uses the camera matrix and distortion coefficients to undistort the image (using OpenCV)
* Returns the undistorted image

setImage(self, cv\_image, undistort=True)

* Sets the processed Image to be cv\_image. This image is undistorted is undistort is True.

boxCordsToPose(self, box)

* Function used to convert the provided box cords (corners of the detected block) to real world coordinates by linearly scaling the image and pixels\_per\_meter value

findBlock(self, color)

* Main function used to find a block of a specified color. These colors are a string used to index the COLOR\_RANGES dictionary which contains HSV ranges for each block color to be detected. Currently contains PURPLE, ORANGE, and GREEN
* This function outputs testCOLOR.png images and displays the image with block squares drawn on them onto baxter’s xdisplay by calling self.displayImage().
* Returns the list of blocks (Block objects)

writeImage(self, img=None, fname=None)

* Writes a cv\_image (img) to a file (fname, default is ‘test.png’)

displayImage(self, img, encoding)

* Displays an image on Baxter’s xdisplay.
* This function is used by findBlock() to display the image with detected blocks

inRange(self, img, low, high)

* Detects the blocks in the desired range using cv2.inRange but also allows for negative numbers so that they wrap and bitwise\_or the results to combine them.

**Block Class**

**Attributes**

color – A string of the block’s color

pose – The pose that this block is located at

coord –The image coordinates of the corners of the block.