University of Glasgow | School of Computing Science

Baxter the Robot - Softness Detection

Craig Hamill

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

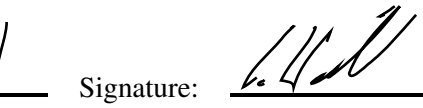
Level 4 Project — March 21, 2017

Abstract

This paper documents how we designed and implemented a system which allows a Baxter Research Robot to detect object locations in 3D-Space and interact with them to perceive a measure of softness using feedback form a low cost, simple gripper.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: Craig Hamill Signature: 

Contents

Chapter 1

Introduction

1.1 Overview

In this paper we present an approach to gaining of measuring the softness from objects through the use of position and load feedback from servos such as the Dynamixel MX-64T [3] used in the EZGripper [13]. We show how we obtain this measure using Hooke's Law and how we have implemented a software system for Baxter the Research Robot [11] which has enabled us to demonstrate, test, and evaluate its usefulness and reliability.

1.2 Aims

The aim of this project is to investigate the possibility of obtaining a measure of softness for a selection of household objects through the use of simple lowcost grippers like the EZGripper [13]. To evaluate and demonstrate this functionality we build a system which allows Baxter to do the following:

1. Determine a softness value for objects using feedback from the EZGripper servo
2. Sort detected objects depending on their perceived softness
3. Classify objects as 'hard' or 'soft'

To allow Baxter to complete these tasks we also develop functionality which allows the robot to find object locations and dimensions in 3D-space.

1.3 Motivation

For humans the the way objects react to an exerted force when we grasp them can tell us a lot about their structure. By exerting multiple varied levels of force we can build a perception of how hard, fragile and smooth an object is. This perception then allows us to decide how best to further interact with the object and how we should interact with similar objects in future.

If robots are to act more like humans and complete more human-like tasks, it is essential they can interact with their surroundings in a similar way. Detecting the softness/hardness of objects with robot hands/grippers is a common area of interest for researchers. Much of this research centres around the development of new, more sophisticated tactile sensors [16][5] [8] to allow more humanlike capabilities for robots such as detecting muscle tissue [9] in the field of medecine or more everyday tasks such as buttoning a shirt or picking up a matches [17].

There is also a need however, for Robots used in more labour intense fields e.g warehouse work, to classify objects depending on their perceived softness. Yet, these tactile sensors often do not appear to be durable enough to cope with the heavy workload that would be expected in scenarios such as assembly or material handling production lines. Furthermore, these sensors are often designed for high end robot hands which would not be suitable for this type of work due to their high cost and delicacy.

There is hence, a solution required which enables robots with low cost sturdy grippers to gain a level of haptic information which could allow robots to interact differently with objects dependant on a perceived measure of softness.

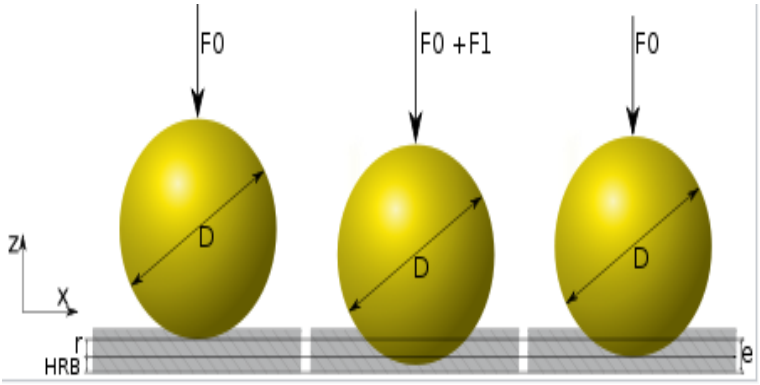
1.4 Background and Approach

For humans to detect the softness of an object we will grasp it with one or both hands then applying a range of forces while observing how much the object yields to each that we have applied. We have therefore, decided this should be the approach taken by Baxter when determining the softness of our objects.

We considered two possible formulas which we might use with this approach to calculate a softness measure using the robots gripper: Hooke's Law and Rockwell Hardness.

1.4.1 Rockwell Hardness

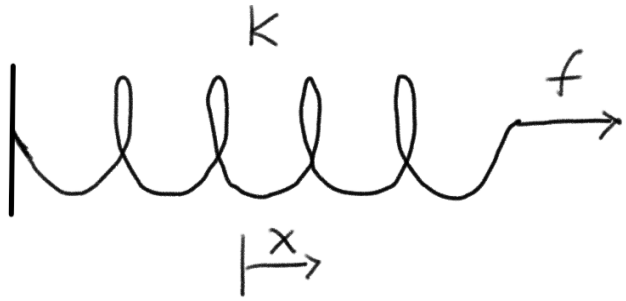
Rockwell hardness is generally used in engineering and metallurgy to calculate and scale the hardness of metals, by measuring the difference in position of an indenter when applying an initial load followed by a final 'major' load. However, we could substitute the depth moved by the indentor for the distance moved by our gripper to give us a measure of hardness/softness for objects as the gripper applies additional force.



The equation for Rockwell Hardness is Harness $HR = N - \frac{d}{s}$ Where d is the depth moved between the two applied forces, and $Nands$ are scale factors which depend on the scale of the test being used.

1.4.2 Hookes Law

Hookes Law is a principal used in physics which states that the force f needed to stretch or compress a spring by a distance x is proportional to that distance i.e. $f = kx$ where k is a constant characteristic of the springs stiffness. It has also been shown to work in many other situations where elastic bodies are deformed [1] [15]. We therefore predicted we could use the k in Hooke's Law as a measure to describe softness/hardness.



1.4.3 Conclusion

While running some initial tests to familiarise ourselves with the Baxter and EZGripper APIs, we noticed the servo would continue moving after the gripper was fully closed or had grasped a solid object. We then observed this was down to the elasticity of the gripper's tendon. As Hooke's law can be extended for two springs in a series, we concluded this would be the best equation to use as it would enable us to remove the movement attributed to the elasticity of the tendon from our final measure.

We have therefore, created a Robot Operating System (ROS) [14] package for Baxter the robot [11] which allows the robot to determine a 'softness' measure for objects, calculated using Hooke's Law from feedback gained from the grippers servo.

1.5 Requirements

1. C++ code which detects the position of objects in 3D space.
2. C++ code which publishes object details to a ROS topic to allow access from the Python code.
3. Python code required for moving Baxter's hand to the desired position to pick up each object.
4. Python code to allow the robot to grasp an object with the EZGripper and algorithm to determine the softness of the item.
5. Python code required for Baxter to re-place items on the table (or move into categorised bins).
6. A ROS launch file which enables the robot to complete its task through the user entering a single command.

Ordering the above requirements using MOSCOW all were categorised as Must Have except the final one (object recognition algorithm) which was categorised as Could Have. Perhaps more important was the order of which each part of the system should be developed. We concluded the above ordering was the most desirable as each item depended on the functionality of its predecessor.

1.5.1 Deliverables

The deliverables for this project will be a presentation demonstrating Baxter accomplishing the task of determining the softness of a collection of objects and a software package which enables this behaviour. The demonstration should include the robot grasping a variety of objects of varying softness and categorising them by placing them in a pre-assigned area or bin depending on the level of softness.

Chapter 2

Materials and Methods

In this chapter we give an overview of the main hardware and software components we have used for this project.

2.1 Baxter the Robot

Baxter is a dual arm robot developed by Sake Electronics. It has gained huge popularity in the manufacturing industry due to its low cost, ability to work around humans without being caged off (like many other manufacturing robots), and ability to be re-programmed to do repetitive tasks by simply moving its joints in the desired fashion. For these same reasons it has also become very popular for use in research over 30 countries around the world [4].

The robot's arms have seven degrees of freedom and can reach up to 1200mm, each with a camera and adjustable parallel grippers on the end point. We have removed the right gripper to attach the EZGripper as it allows much wider objects to be grasped and allows us to get position feedback to use in our softness calculation.

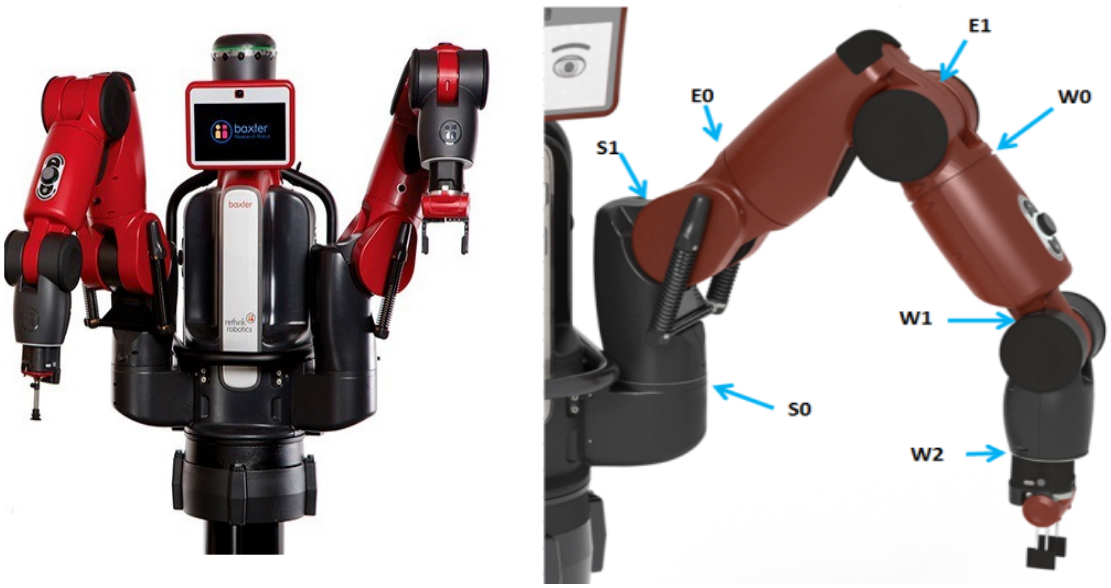
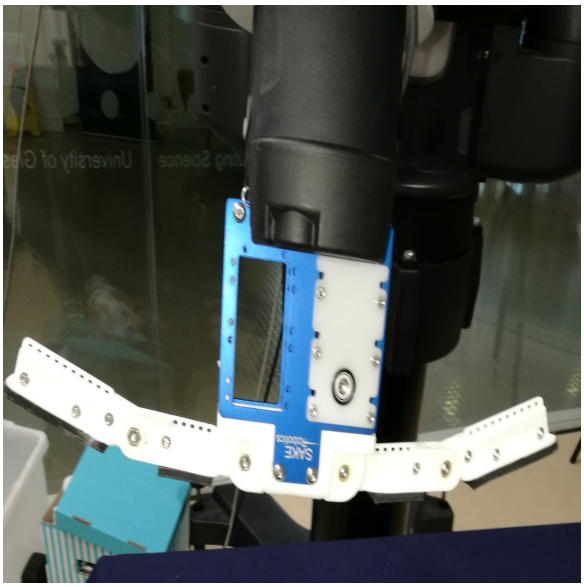
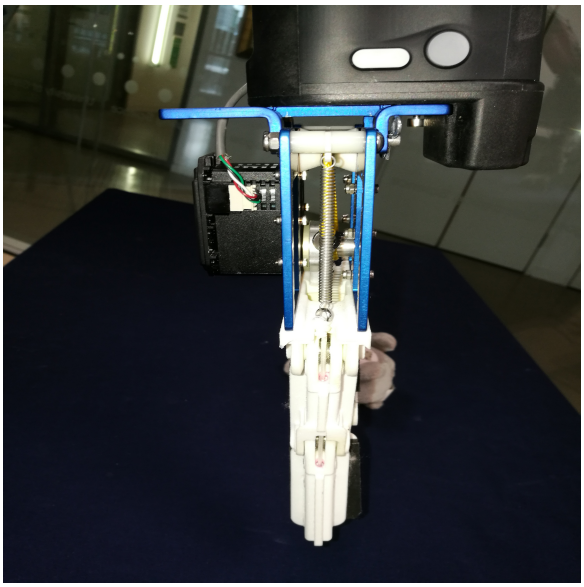


Figure 2.1: Baxter and limb joint positions

2.2 EZGripper

The EZGripper is a two fingered gripper which has a rotational grasp. The two plastic fingers which connect to the gripers lightweight aluminium base, are controlled via a string tendon which is attached to the grippers servo. As each finger has two joints, the rotational grasp allows the gripper to pick up a wide variety of objects upto 170mm in width. The servo used in this gripper provides position and load feedback, which allows us to retrieve the required information for our softness calculations.





2.3 Robot Operation System (ROS)

The ROS is an open source framework which provides operation system functionality to robots such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.It also contains tools, libraries and conventions for developing robust robot behavioural systems.

Within ROS, each executable file/node is loosely coupled to other nodes on a peer-to-peer network through the use of the ROS communication infrastructure. As it has been built from the ground up to encourage collaboration and centres on publish/subscriber and request/response relationships between nodes, ROS enables developers to concentrate on a particular part of the system which is relevant to their research, while using packages shared by others to do everything else. This approach has led to ROS becoming the go-to framework for robot software development and is now used in some of the wolds best known robots such as Baxter and the PR2 [11][2].

As both the Baxter and EZGripper interfaces are packaged in ROS packages, developing our software using ROS allows us to interact with both through services and topics. It also provides us with all of its operating system like features and allows us to access other packages which we require for our project such as PCL and the glasgow_calibration package [6].

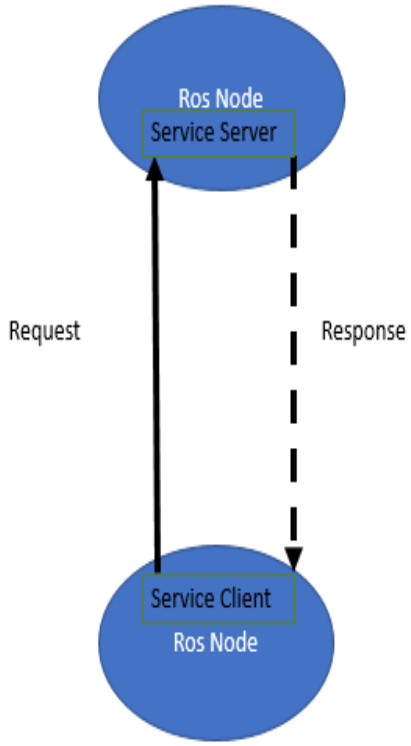


Figure 2.2: Example ROS Publish - Subscriber relationship

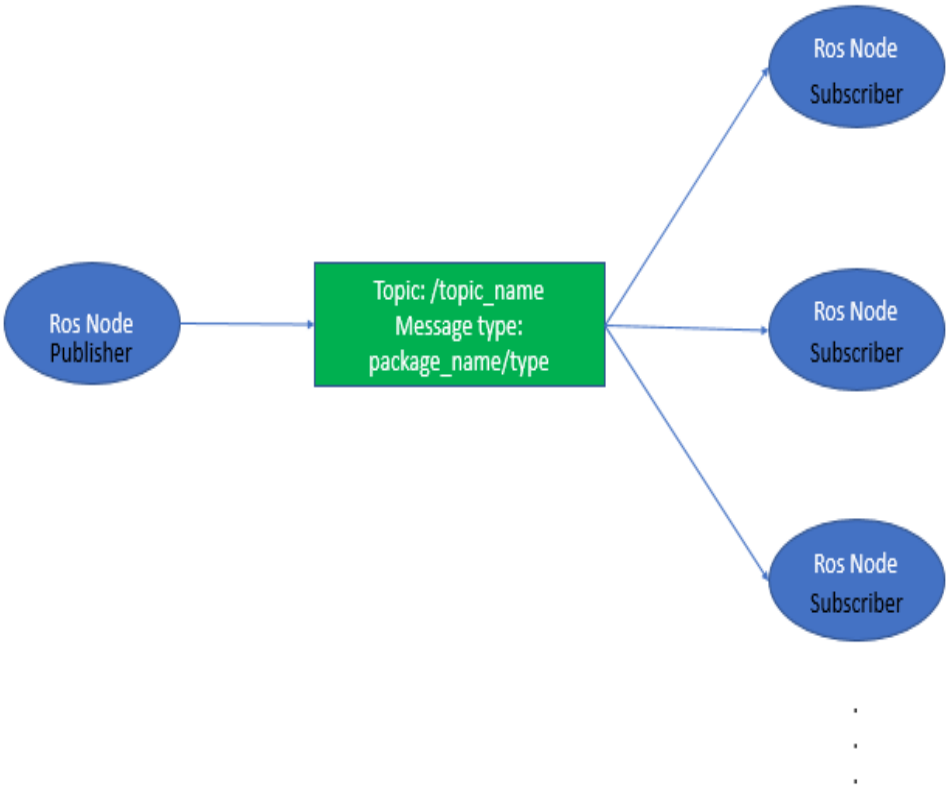


Figure 2.3: Service server - Service Client Relationship

We will go into more detail on ROS and these relationships as we describe our implementation in the next chapter.

2.4 Point Cloud Library (PCL)

We considered using OpenCV with images taken from Baxter's hand mounted camera or PCL with a depth sensor such as the Kinect for detecting the location and dimensions of objects. Bellow we list the advantages/disadvantages of each approach, before explaining why we arrived at our final decision to use PCL.

2.4.1 OpenCV

Benefits

1. Lots of documentation on functionality for detecting objects in images.
2. No need for additional camera as we can use built in camera on robots hands.

Disadvantages

1. To calculate height/width of objects need to have an object of known size in range of camera during each execution of program.

2. Difficult to determine depth/height of objects (depending on camera orientation)

2.5 Kinect Sensor with PCL

2.5.1 Benefits

1. Easy to learn and use functionality for detecting object locations in 3D-space.
2. Once a cluster representing object is detected, all 3D-points within it are real world coordinates relative to the position of the sensor. A static transform can then be used to translate the coordinates such that they are relative to the position of the robot.
3. Has a lot of algorithms for segmenting objects into clusters which can then be used to calculate features such as height width and depth.
4. Clusters can be saved to point cloud files to use with built in object recognition functionality (if to be added later).

2.5.2 Disadvantages

1. Requires Kinect sensor to be calibrated with the robot to transform points relative to the sensor to make them relative to the robot.
2. Must recalibrate sensor if robot or sensor is moved.

We chose to use the Kinect sensor together with PCL for locating the location of objects due to the relative simplicity in detecting the their position and dimensions. Additionally by using a Kinect sensor we can process the scene at any time and are not limited to only doing so when the robot's hand is in a certain position, as we would have been if using OpenCV.

Chapter 3

Implementation

Here, we detail how we implemented our system by creating two new ROS packages and making some slight modifications to the EZGripper driver. First we explain the relationships between our nodes within these packages before describing the relationships between these nodes and others within packages which we have not added or altered in anyway. We will then describe our code and the algorithms we have used to enable the robot to sort objects depending on their softness.

3.1 Node Relationships

As briefly described earlier ROS nodes communicate in one of two ways: topics or services.

3.1.1 Topics

Communication through topics is done via publish / subscribe semantics. One or many publishers can publish a particular message type to a topic while one or more subscribers subscribe to a topic which receives a type of data they are interested in. This decouples the publisher and subscriber as neither need to know of the others existence all that each requires the name of the topic they wish to use.

3.1.2 Services

The publish/subscribe semantics of topics is very flexible. However asynchronous streaming of data is often not what we require e.g. when our application first starts we want to get the location of objects as soon as possible so we can start moving the robot's hand to the fist object's position. Using a topic we had to guess how long the cluster detection node would take to detect the objects and publish their dimensions etc. before trying to retrieve the data from the topic. This would cause an observable delay as we had to over estimate the time needed to retrieve the object details to avoid retrieving null values.

Services provide a synchronous Remote Procedure Call (RPC) like functionality which allows us to avoid this problem as our node will send a request to the service, then wait for the reply before continuing execution. As services work much like RPCs they also allow parameters to be passes through messages trough the request.

3.1.3 Altered and Created nodes

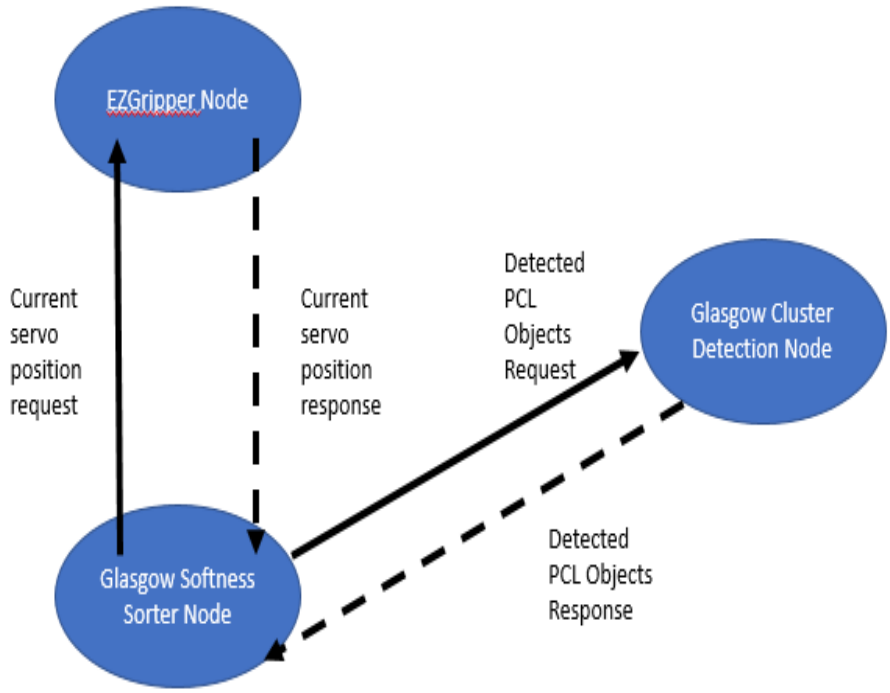


Figure 3.1: Relationships between created or altered nodes

The Glasgow Softness Detection node is the class we use to calculate a softness measure for each object. This node issues a request to the EZGripper node when it wishes to retrieve the current position of the servo. It then uses this position in the softness calculation. The EZGipper.py file is supplied within the EZGripper package which is used to control the gripper. We did not alter this package in any other way than to add the service to return the servo position.

The message returned form the service is of type: $ezgripper\_driver/current\_servo\_position$, which contains one 32 bit integer: $current\_servo\_position$. We used a service in this case instead of a $publish/subscriber$ relationship as we did not want to add the extra overhead to the EZGripper driver of constantly publishing the servo position to a topic. Using a service also allows the calling node to wait on the response before continuing,which we found to be beneficial as our next command was often to move the robots hand in a manner which depended on the calculated softness value.

The Softness Sorter node is our main class which is used to sort items depending on their softness. This node uses a SoftnessDetector instance to retrieve softness values for each object being sorted. To instruct Baxter to grasp these objects it must first acquire the dimensions and positions of each object. This is done through the $glasgow\_object\_detection/detected\_pcl\_objects$ service made available by the Cluster Detection node.

This service takes no arguments and returns an array of $glasgow\_object\_detection/grasp\_position$ messages, each containing 32 bit integers representing the x,y, and z coordinates of the objects centroid, the object's width height and depth, and the maximum z point of the object.

3.1.4 External Nodes

External Topics

We topics which require three nodes and one launch file which are external to either of our packages to launch on start up. They are therefore included in our softness_detection.launch file. Three of these are included in packages which are supplied with Baxter to enable certain functionality.

1. baxter_tools/enable_robot
2. baxter_examples/xdisplay_image
3. baxter_interface/gripper_acction_server
4. kinect2_bridge/launch/kinect2_bridge.launch

The first two do not require further explanation other that to state the first enables the robot by supplying power to the joint motors, the second is used to display an image on the head mounted screen and the third is used to control the default gripper on Baxter's left hand. Our softness sorter uses this gripper while sorting items using the bubble_sort function. The kinect2_bridge however, requires a little more detail.

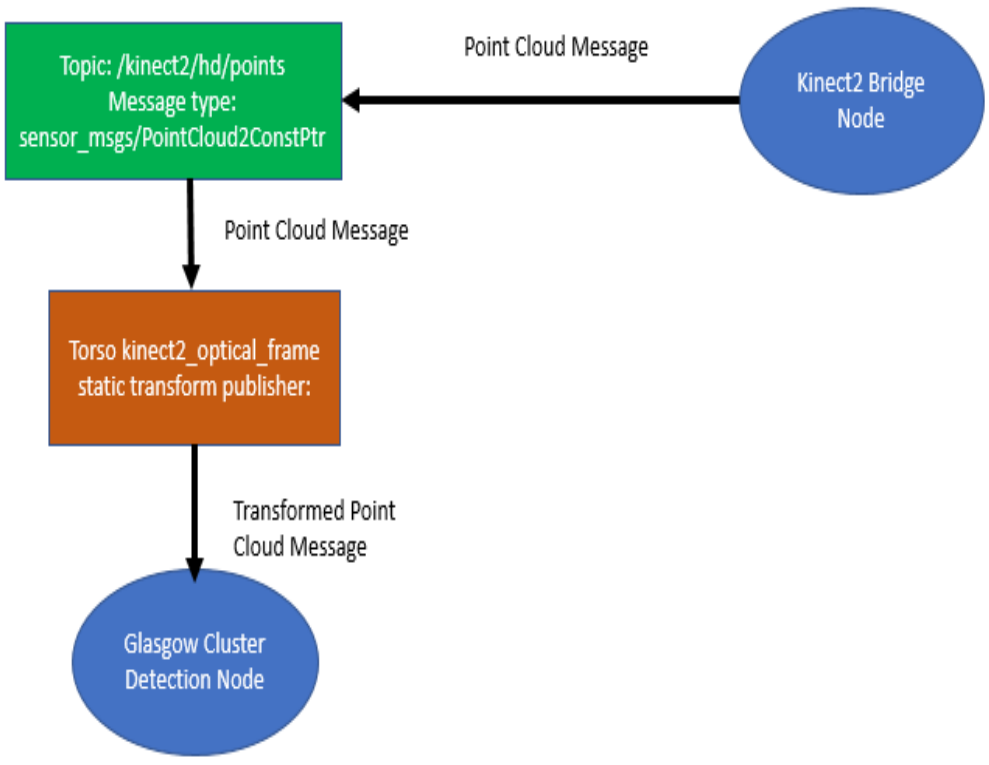


Figure 3.2: Cluster Detection and /kinect2/hd/points Relationship

Our Cluster Detection node requires depth information gathered from the kinect sensor in the form of point clouds. To this end we use the iai_kinect2 package [18] which publishes point clouds and other information

to several topics. As point cloud coordinates are given relative to the Kinect sensor we must first use a static transform to translate them to coordinates relative to the torso frame of Baxter.

To set up this transform we use the Calibration Glasgow package [6] to find the pose of the Kinect relative to the torso. This requires us to make Baxter grasp a wooden board with black and white chess board like grid on the front. We must then capture ten images through the Kinect via the calibration program, each with the holding limb joints in different positions. The program then calculates and outputs the camera pose required to add to the static transform publisher which is created in our launch file.

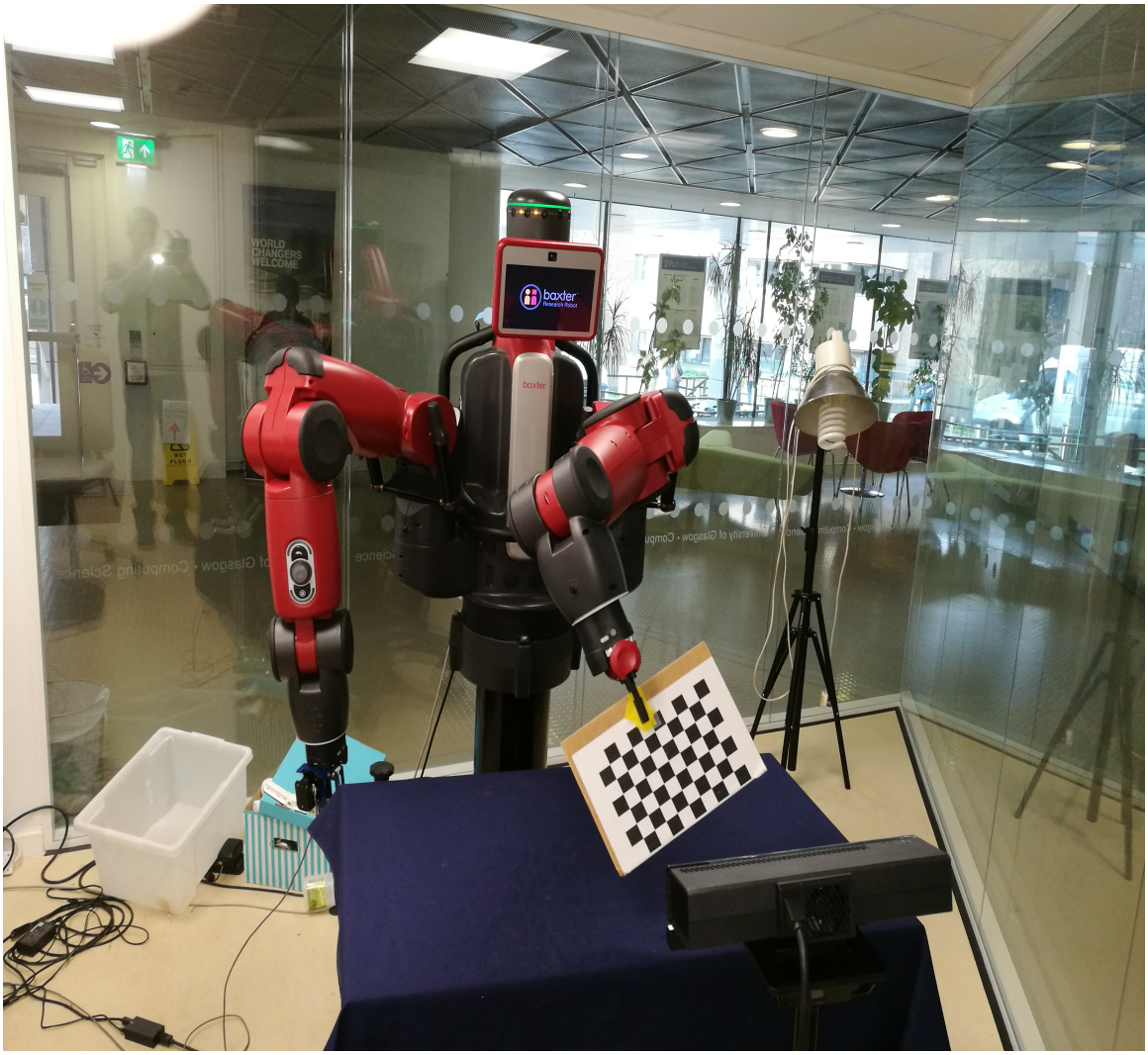


Figure 3.3: Baxter holding board used for Kinect calibration

We have found the pose acquired from this package to be a little unpredictable; occasionally it gives coordinates which are almost perfect i.e. when we direct the robot to pick up an object it will go to the correct position to grasp it. However, we have found that it often gives us coordinates which are a little off e.g. for our fourth experiment we found they were around 20mm less than what they should be on each axis.

This small error appears to be approximately a constant, which can be found by placing two objects at different positions on the table then commanding the robot to move to the location to grasp them. We then calculate a rough mean of how far the gripper is from the desired grasp point on each axis. These values can then be added or subtracted from the values for each axis when the robot is commanded to move the gripper to a new location.

We have however, only observed the yaw. pitch or roll to be noticeably inaccurate on one occasion. Therefore if this happens we advise the calibration should be redone to avoid additional configuration.

External Services

In addition to the dependencies on topics published to by the above nodes our Softness Sorter node depends on the ROS $ExternalTools/'limb'/PositionKinematicsNode/IKService$ services (where limb = 'right or 'left' depending on lib being used). These services are used to convert Cartesian space representations of the desired arm endpoint (x,y,z, roll, pitch, yaw) to seven degrees of freedom joint states (given in radians for each joint angle).

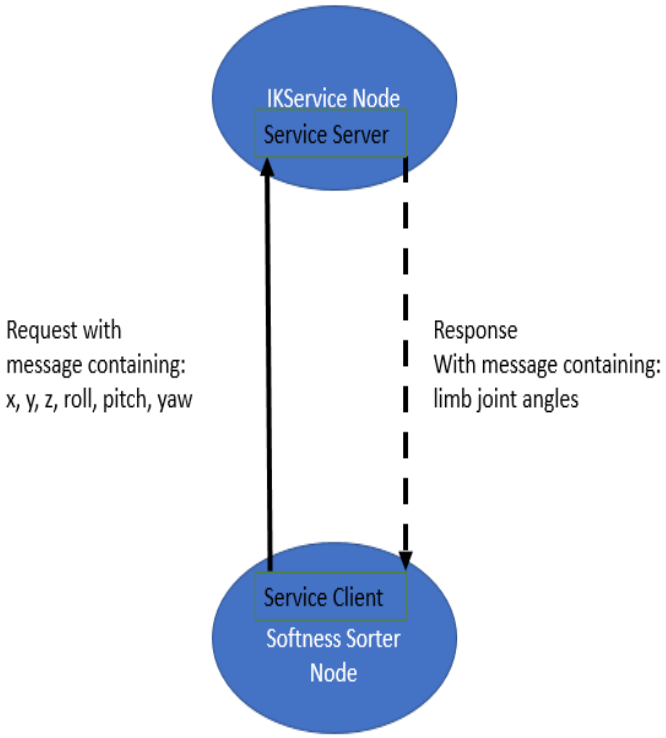


Figure 3.4: Softness Sorter and Inverse Kinematics Service Relationship

Once we have retrieved joint positions for a limb we can check if they are valid before instructing the Baxter interface to move the joints, as the service also returns a boolean value $isValid$ in addition to the joint angles. Joints are moved by passing the joint angles to the $move\_to\_joint\_possitions()$ function in an instance of the Limb class which is imported from the $baxter\_interface$ package. We have found that this method will cause the arms to move in a strange, un-human-like manner if it is instructed to move the limb to a position which is significantly different from its current state.

To avoid this we move the limb to a midpoint between our desired positions e.g. if we wish to move the right hand from its default position (which is slightly raised from its relaxed position at the robot's side) to the location of an object on the table, we will first move the hand to a position above the robot with the same z coordinate as its current position.

3.2 Created Packages and Code Implementation

3.2.1 Glasgow Object Detection Package

The $glasgow\_object\_detection$ package is where we have implemented the code required to detect object locations. This is done within a single c++ file: $cluster\_detection.cpp$ consisting of a main function which runs when the node is started and a $get\_objects$ function which is attached to the $detect\_pcl\_objects$ service. This has one empty request argument and returns a response containing an array of $object\_location$ messages.

When the main function is first called, it subscribes to the '/kinect2/hd/points' topic then advertises the $detect\_pcl\_objects$ service (which calls the $get\_objects$ function). When the $get\_objects$ function is called it first grabs a point cloud object from the $kinect2$ topic then uses the static transform publisher to make the point cloud's Cartesian coordinates relative to Baxter's torso frame. Giving each point's x, y and z coordinates in meters from thee torso's origin.

Area filtering

Once the point cloud is retrieved and transformed we remove all points which are outside our normal working area which is 800mm long by 400mm wide rectangle on the table in front of Baxter, centred on the torso's y origin, 500mm from the origin on the x axis. This vastly reduces the computations required for the subsequent steps.

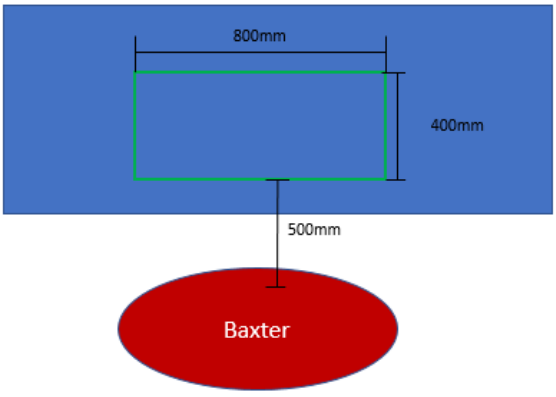


Figure 3.5: Working Area

This is achieved using the following code:

```
// build the condition to remove any points outside the reach of both hands
pcl::ConditionAnd<pcl::PointXYZ>::Ptr range_cond(new pcl::ConditionAnd<pcl::PointXYZ>());
range_cond->addComparison(pcl::FieldComparison<pcl::PointXYZ>::ConstPtr(new pcl::FieldComparison<pcl::PointXYZ>(
        "x", pcl::ComparisonOps::GT, 0.5)));
range_cond->addComparison(pcl::FieldComparison<pcl::PointXYZ>::ConstPtr(new pcl::FieldComparison<pcl::PointXYZ>(
        "x", pcl::ComparisonOps::LT, 0.9)));
range_cond->addComparison(pcl::FieldComparison<pcl::PointXYZ>::ConstPtr(new pcl::FieldComparison<pcl::PointXYZ>(
        "y", pcl::ComparisonOps::GT, -0.4)));
range_cond->addComparison(pcl::FieldComparison<pcl::PointXYZ>::ConstPtr(new pcl::FieldComparison<pcl::PointXYZ>(
        "y", pcl::ComparisonOps::LT, 0.4)));

// build the filter
pcl::ConditionalRemoval <pcl::PointXYZ> condrem(range_cond);
condrem.setInputCloud(tf_cloud);
condrem.setKeepOrganized(true);

// apply filter
pcl::PointCloud<pcl::PointXYZ>::Ptr dist_filtered(new pcl::PointCloud <pcl::PointXYZ>);
condrem.filter(*dist_filtered);
```

Figure 3.6: Area filter code

Segmentation

We then further downsample the point cloud using a voxel grid with leaf size 10mm x 10mm x 10mm. Clusters representing our objects are then extracted using the following algorithm:

1. Create a Kd-tree representation for the input point cloud dataset P;
2. Set up an empty list of $detected\_object$ messages D, and a queue of the points that need to be checked Q;
3. For every point $\boldsymbol{p}_i \in P$, perform the following steps:
   - Add $\boldsymbol{p}_i$ to the current queue Q;
   - For every point $\boldsymbol{p}_i \in Q$ do:
     - search for the set P_k^i of point neighbours of $\boldsymbol{p}_i$ in a sphere with radius $r < d_{th}$ (where depth d_{th} is set to 20mm);
     - For every neighbour $\boldsymbol{p}_i^k \in P_i^k$, check if the point has already been processed, and if not add it to Q;
   - When the list of all points in Q has been processed, Q now contains a cluster of points representing one of our objects.
   - From Q add the centroid, height and maximum coordinate on the z axis to a $detected\_object$ message and add to D.
   - Reset Q to an empty list
4. The algorithm terminates and returns D in service response when all points $\boldsymbol{p}_i \in P$ have been processed.

This algorithm and the code implementing it are modified versionsof the PCL 'Euclidean Cluster Extraction' tutorial [10]. We have changed the minimum cluster size, removed some of the output code, and added the steps for creating/returning the array of $detected\_object$ messages in the service response. We found this algorithm to give good results, allowing us to concentrate effort on other parts of our system. The minimum cluster size is set to 100 in the tutorial however, we found that our small wooden block would not be detected with this setting so reducer the minimum size to 30, as we found this worked without adding any noise to our cluster set.

This algorithm allows us to find our object locations and dimensions fast because of the use of a Kd-tree, which is a k dimension binary tree (where k is set to 3 as we are using three dimensions) which branches at each level depending on a different axis. Building a kd-tree with our remaining filtered points allows us to find the nearest neighbours of each point in almost logarithmic time. Building a point cloud kd-tree is also very simple using PCL and can be done using only two lines of code:

```
// Creating the KdTree object for the search method of the extraction
pcl::search::KdTree<pcl::PointXYZ>::Ptr tree(new pcl::search::KdTree <pcl::PointXYZ>);
tree->setInputCloud(cloud_filtered);
```

Figure 3.7: kd-tree code

3.2.2 Glasgow Softness Detection Package

The $glasgow\_softness\_detection$ package contains all our functionality for controlling Baxter and allowing objects to be sorted by their perceived softness. This package contains three python files/classes: $softness\_sorter$, $softness\_detection$ and $softness\_object$. The $SoftnessSorter$ class is our main class which contains the functionality for sorting objects. This class uses instances of $SoftnessObject$ to hold details of objects acquired through the $detect\_pcl\_objects$ service and uses an instance of the $SoftnessDetector$ class to obtain a softness measure for each object using the EZgripper.

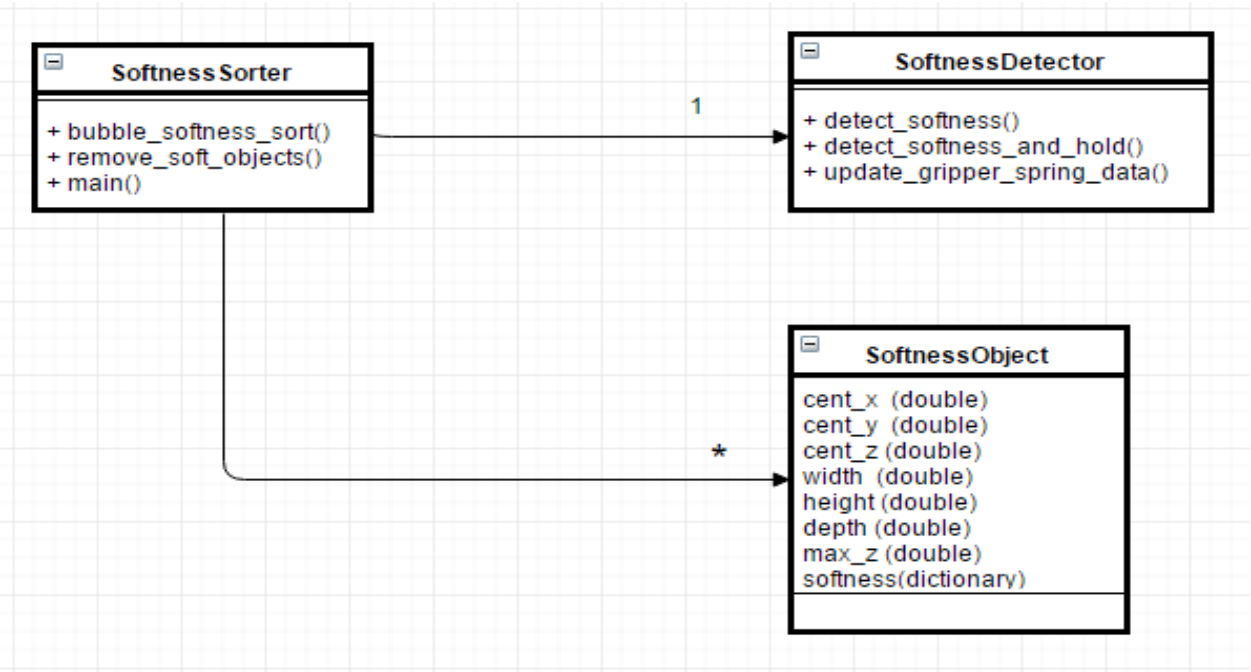


Figure 3.8: Softness Detection Class Diagram

The $SoftnessObject$ class only contains the listed variables and no methods as the instances are only used

to store information about the objects being manipulated. $SoftnessSorter$ and $SoftnessDetector$ do however, contain lots of other helper functions and variables but we feel the ones listed in the diagram are the ones needed to describe our implementation.

We have created two functions for Baxter to demonstrate how we have enabled softness detection through the use of Hooke's law and the feedback from the gripers servo. We will first explain how our softness values are calculated using the $softness_detection$ class within this package, before giving an overview of our two sorting algorithms which are implemented within the $SoftnessSorter$ class.

3.2.3 Softness Detection

The $SoftnessDetecter$ class is used to control the EZgripper and calculate softness measures for objects using Hooke's Law and feedback obtained from the gripper servo.

Algorithm

Hooke's Law gives us a value which decreases the more the object's surface can be manipulated given an applied force, we have therefore chosen to use $softness = \frac{1}{k}$ as this gives a value which increases as the surface is manipulated and approaches zero the more resistant it is to the applied force.

We used Hooke's Law for to springs in a series where the grippers tendon is treated as spring $s1$ and the objects surface spring $s2$. This gives $\frac{f}{k} = \frac{f}{k1} + \frac{f}{k2}$, where k is the effective spring constant, $k1$ is the spring constant of the gripper tendon, $k2$ is the spring constant of the objects surface and f is the applied force.

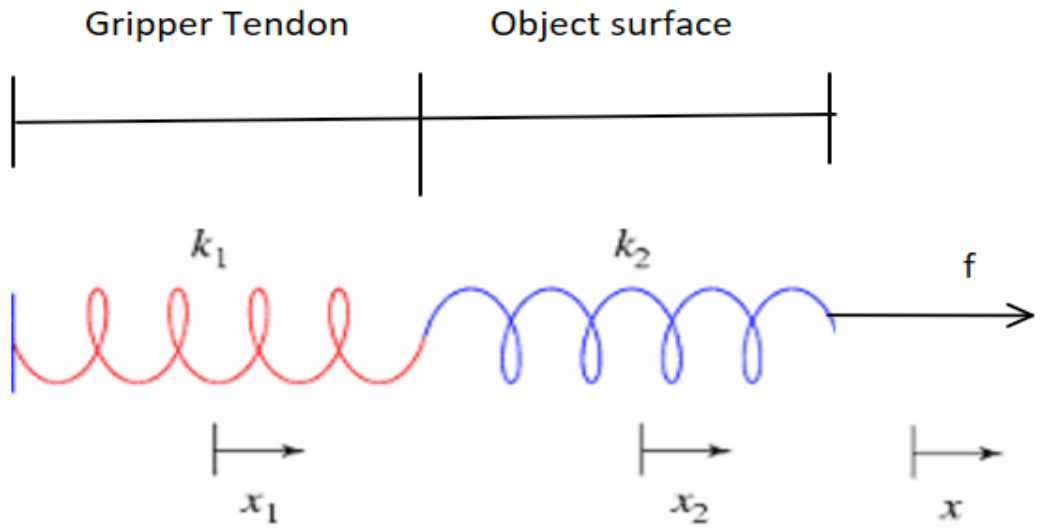


Figure 3.9: Application of Hooke's Law

As Hookes Law also gives $\frac{f}{k} = x$, where x = distance moved, given that we want to obtain the value of $k2$, we can rearrange this formula to give $k2 = \frac{f}{x-x1}$, where $x = \frac{f}{k}$ = total distance moved between p_{start} and current position $p_{current}$, and $x1 = \frac{f}{k1}$. Our softens value calculated after applying each force is then $softness = \frac{1}{k2}$.

During our experiments the EZGripper is instructed to go to the fully closed position with a fixed maximum effort while grasping each object. As the gripper will not reach this position given the object's width is > 0, the servo will turn until the maximum effort/load is reached. We therefore use this setting as the force in our softness calculation and will refer to it as force or f throughout this paper.

Code Implementation

The main functions of this class are: $detect\_softness\_and\_hold()$ and $update\_gripper\_spring\_data()$. The latter is important as it is used to calculate $x1$ for each force ranging from 5% to 100% (in steps of 5%) and save to disc. The data is then loaded from disk and stored to the $GRIPPER\_SPRING\_MOVEMENT$ dictionary where each applied force maps to its $x1$ value. each time the _init_() method of the class is called. The method will automatically be called if the output file is empty, but should also be called once every two weeks of gripper use as the tendon may loose some of its elasticity.

The $detect\_softness\_and\_hold()$ method is the most important in this class as it is used to instruct the gripper to grasp an object and returns the perceived softness value (note there is also a similar detect object function, however, the only addition to this function is that the gripper is opened after detecting the object's softness). The method, which takes one argument (the effort/force the gripper should stop moving at) first closes the gripper with $f = 5\%$ which causes the gripper to stop moving when it comes into contact with the object's surface. The $current\_servo\_position$ service is then called to retrieve the servo position after waiting 3 seconds to ensure the gripper has sufficient time to reach the point when effort reaches 5%. The helper function $\_get\_softness()$ is then called, passing the current servo position and the effort/force f which softness should be detected at.

The softness value is then calculated using equation stated earlier in the following code:

```
# method detects softness value using hooke's law
def __get_softness(self, first_grasp_pos, effort):
    self.ez.close(effort)
    rospy.sleep(self.TIME_BETWEEN_MOVEMENTS)

    distance_moved, end_pos = self.__get_gripper_displacement(first_grasp_pos)
    rospy.loginfo("distance moved: " + str(distance_moved) + ", string movement: " + str(self.GRIPPER_SPRING_MOVEMENT[str(effort)]))

    distance_moved -= self.GRIPPER_SPRING_MOVEMENT[str(effort)]
    softness = 0
    if distance_moved != 0:
        k = float(effort) / (float(distance_moved))
        softness = 1/k
        return softness

    return softness
```

Figure 3.10: Application of softness detection algorithm

The other methods in this class are mostly wrappers for the EZGripper API functions with the addition of returning the distance moved by the servo after each action.There are also however, two methods which can be used to acquire a range of softness values calculated for either a number of grasps with the same applied force ($test\_softness()$), or a number of forces over the same grasp ($test\_softness\_range()$). These methods were used during our first two experiments to gather data but have no use in our $SoftnessSorter$ application.

3.2.4 Softness Sorting

The $SorftnessSorter$ class contains the main functionality of our application. It utilises the $SoftnessDetector$, $SoftnessObjects's$, the $detect\_pcl\_objects$ service and the $inverse\_kinematics$ service together with the Baxter API's, to allow Baxter to either sort items by softness, or remove items from the table which are classed as 'soft'.

Softness Bubble Sort

The $softness\_bubble\_sort()$ method sorts upto four items depending on their measured softness using a reverse bubble sort algorithm where the softer items will be moved to the left while the hardest item in each iteration will be moved to its rightful place at the $n-j-1$ location. When the method is first called the $get\_object_{l}ocations()$ method is called to retrieve the object list acquired from the $detect\_pcl\_objects()$ service are sorted by their centroid's y axis value. This is done so Baxter will pick up the objects starting in the leftmost position. Baxter will then grasp the first object using the EzGripper to detect its softness before releasing its grasp and moving to the waiting position to begin sorting using the following code /algorithm:

```
swapped = True
# loop until we no objects are swapped during an iteration as this indicates the
# objects are in correct order
j = 0
while swapped:
    swapped = False
    # loop through  each pair of adjacent objects and swap  if the softest object is not in the
    # left gripper
    for i in range(1, len(self.detected_objects) - j):

        co = self.detected_objects[i]
        po = self.detected_objects[i - 1]

        # pickup objects if current object is softer than previous object
        # or if softness class = '' as this indicates  co's softness has not yet been detected
        if co.softness['softness_class'] == '' or self.is_softer(co, po):
            result = self.pickup_using_right_gripper(co)
            if result != 0:
                rospy.sleep("Cannot reach object with right gripper!")
                return -1
            rospy.sleep(self.DEFAULT_SLEEP_TIME)
            result = self.pickup_using_left_gripper(po)
            if result != 0:
                rospy.sleep("Cannot reach object with left gripper!")
                return -1

            # swap if current object is softer than previous objects,
            # moving the harder object right.
            if self.is_softer(co, po):
                swapped = self.swap_objects(co, po, i)

            result = self.place_using_left_gripper(po)
            if result != 0:
                rospy.sleep("Cannot reach location to place object with left gripper!")
                return -1
            rospy.sleep(self.DEFAULT_SLEEP_TIME)

            result = self.place_using_right_gripper(co)
            if result != 0:
                rospy.sleep("Cannot reach location to place object with right gripper!")
                return -1
        rospy.sleep(self.DEFAULT_SLEEP_TIME)
        j += 1
```

Figure 3.11: Softness Sorting Algorithm

This sorting algorithm is a relatively simple implementation of Bubble Sort, with the addition of the checks due to the possibility the object location might be out of reach of the gripper. However there is also a lot of steps required when picking or placing objects e.g. the steps required to pick up one object (similar steps are also required when placing):

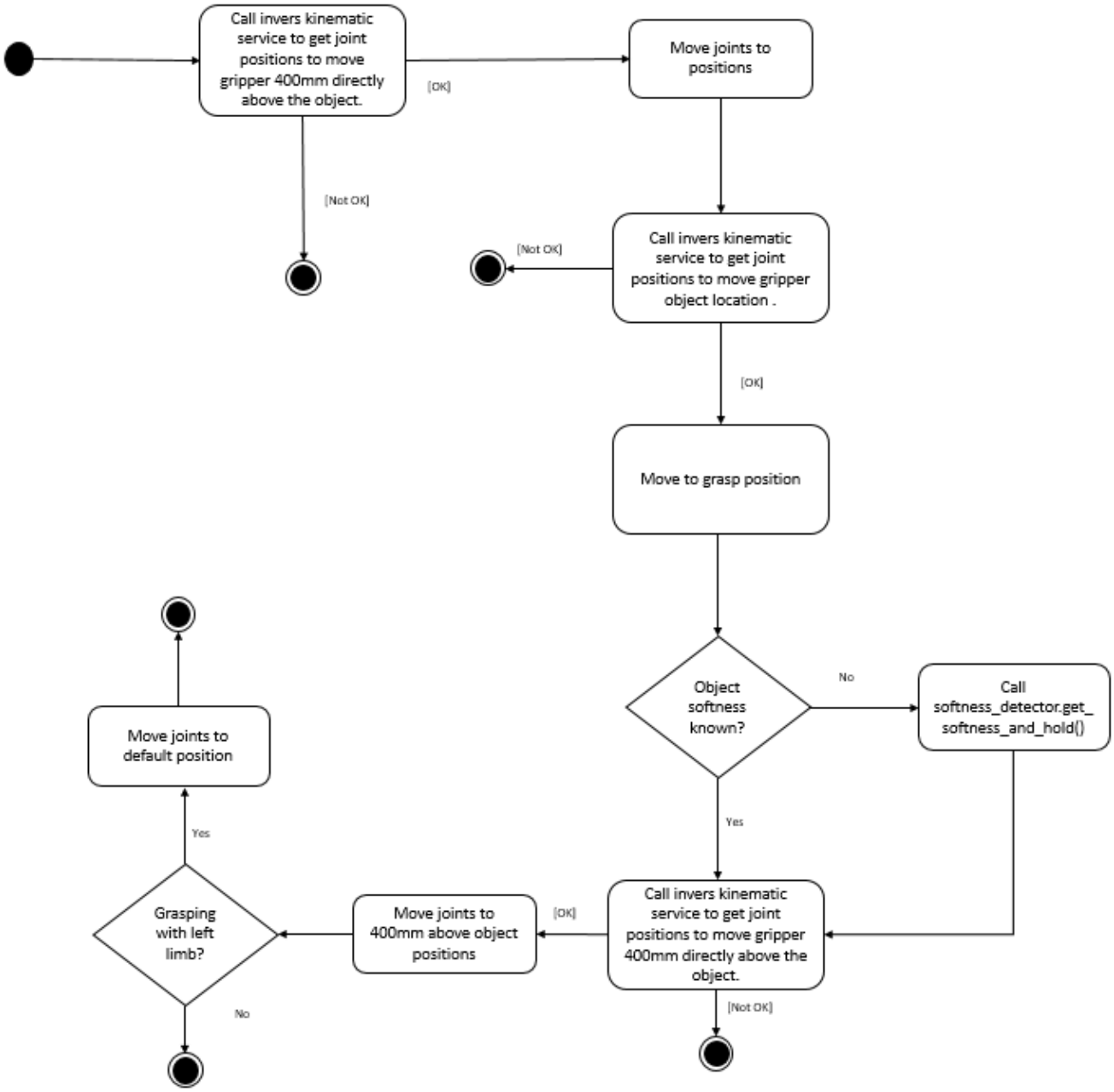


Figure 3.12: Activity Diagram for $pickup\_using\_left/right\_gripper()$

In each iteration the softness of the ith object is compared to the $i-1th$ object; if it has a higher softness value, or if the softness for the ith object has not yet been observed, Baxter will grasp the objects with the right and left grippers respectively and move the gripers to a waiting pose directly above the point where the objects were grasped. If the ith object is perceived to be the softer of the two, Baxter will then swap the positions of the objects when placing. Otherwise they will placed back in the positions they were found.

Soft Item Removal

When the $move\_soft\_objects\_to\_bin()$ method is called the $get\_object_{l}ocations()$ method is called as before in the bubble sort method. The list does not require sorting in this case though as the items can be grasped in any order. Once the object list is returned Baxter will grasp each object to determine its softness class, which is either $hard$ or $soft$ and is assigned to each object when its softness is first detected depending on the value of the object's $HARD\_LIMIT$ variable i.e if $softness > HARD\_LIMIT : class = soft$ else $class = hard$. We have currently set $HARD\_LIMIT$ to 1.8 as this is the midpoint between the minimum softness attributed to a $soft$ item and the maximum softness value attributed to a $hard$ item in our first two experiments.

This function is implemented using the following code/algorithm:

```
# loop through detected objects
for i in range(0, len(self.detected_objects)):
    co = self.detected_objects[i]
    result = self.grasp_using_right_gripper(co)
    #  check if grasp was successful before continuing
    if result == 0:
        rospy.sleep(self.DEFAULT_SLEEP_TIME)
        arm = baxter_interface.Limb(self.limbs[1])
        # if object was classed as hard let go and move to waiting pose
        # else move object to bin
        if co.softness['softness_class'] == 'hard':
            self.sofness_detector.open_gripper()
            rospy.sleep(self.DEFAULT_SLEEP_TIME)
            self.move_to_waiting_pose(co, self.right_above_obj_z, self.limbs[1])
            rospy.loginfo("Hard object")
        else:
            self.right_holding = True
            rospy.sleep(self.DEFAULT_SLEEP_TIME)
            arm.move_to_joint_positions(self.ABOVE_BIN_POS)
            rospy.loginfo("Soft object")
        # no need to check result of above action as gripper still needs to open
        rospy.sleep(4)

        self.sofness_detector.open_gripper()
        rospy.sleep(self.DEFAULT_SLEEP_TIME)

self.right_arm.move_to_joint_positions(self.RIGHT_START_JOINT_POS)
```

Figure 3.13: Activity Diagram for $pickup\_using\_left/right\_gripper()$

Unlike the sorting method we do not return from the method if an object is in an invalid grasp position When the $grasp_using_right_gripper()$function is called, if the inverse kinematics service judges the object to be in an invalid grasp position w e print this to screen. We feel this is sufficient for this function as we can simply let the user know the object is in an invalid position before moving on to the next. The robot will then stop once all objects within reach have been grasped and all 'soft' objects are removed to the bin position.

Chapter 4

Evaluation

In this chapter, we evaluate our approach to detecting softness. Our first two experiments are designed to prove our hypothesis that we can gain a measure of softness using Hooke's Law; the three that follow are used to evaluate if this approach works in practice using Baxter and our implementation of the the two sorting functions.

4.1 Validation

4.1.1 Experiment 1

Overview

Our goal is to prove our hypothesis that we can use Hooke's Law to obtain a softness measure which could be used to classify objects into two sets: hard and soft.

Test Set

We gathered a training set of six objects split into two sets: three which we have classified as 'soft' S and three 'hard' H. Our aim was to find six household objects which we could split into hard and soft sets with a high degree of confidence. For this reason we chose three items with surfaces which we predicted could not be manipulated with any of the grippers forces, and an additional three which could be manipulated with relatively small force.

The objects used in this experiment are detailed bellow:

Soft Set

Child's soft bear toy measuring around 55mm at grasp point

Child's soft dog toy measuring around 100mm at grasp point

Child's soft cat toy measuring around 200mm at grasp point

Hard Set

Wooden block measuring around around 40mm at grasp point

Metal tea container maesuring around 200mm at grasp point

Metal biscuit container measuring around 200mm at grasp point

Design

During this experiment, the EZGripper is moved to a position such that it is centred on the object and the object can be grasped with the end section of the gripper.

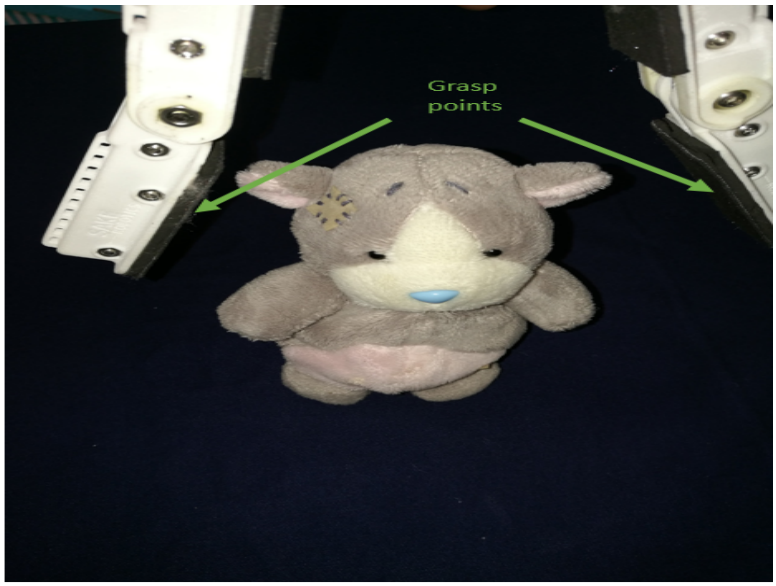
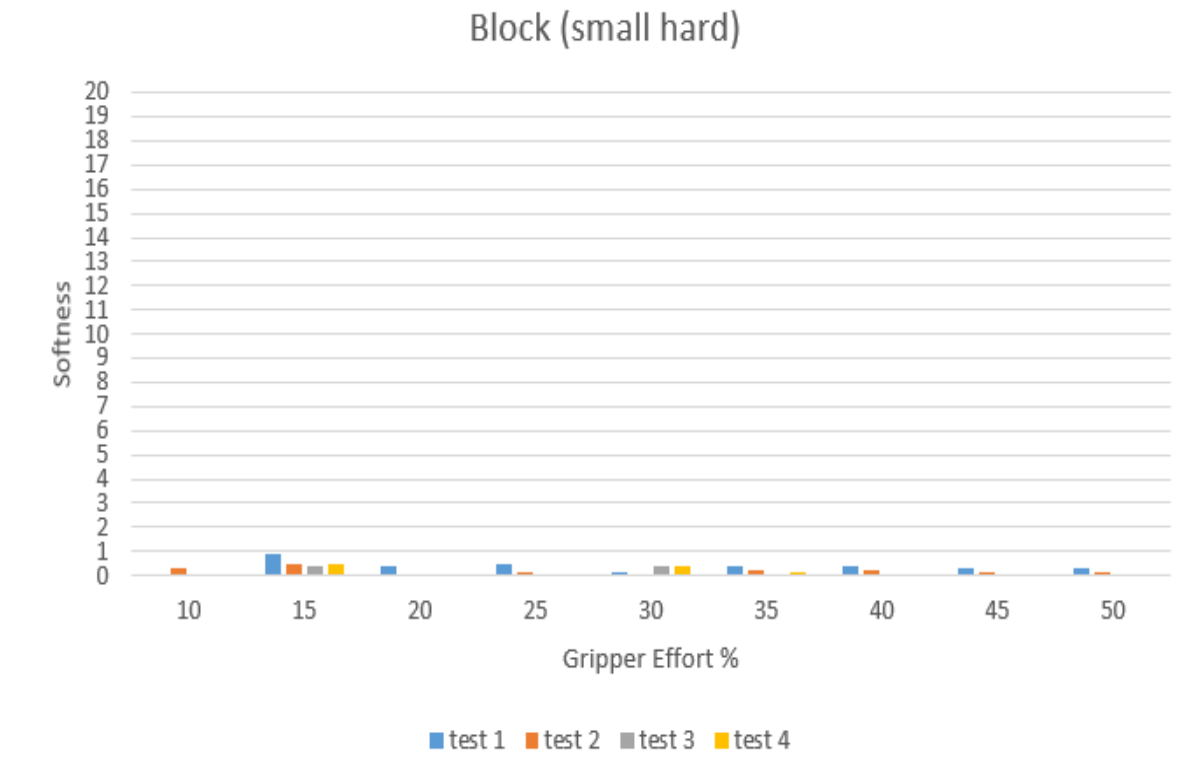
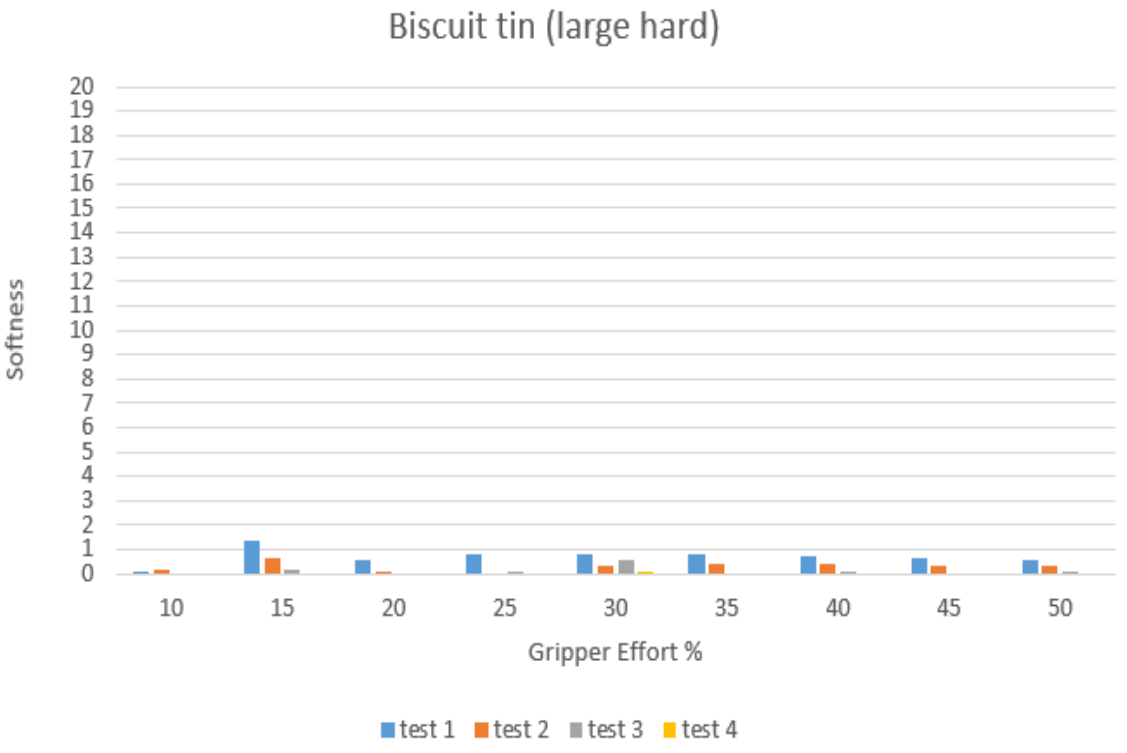


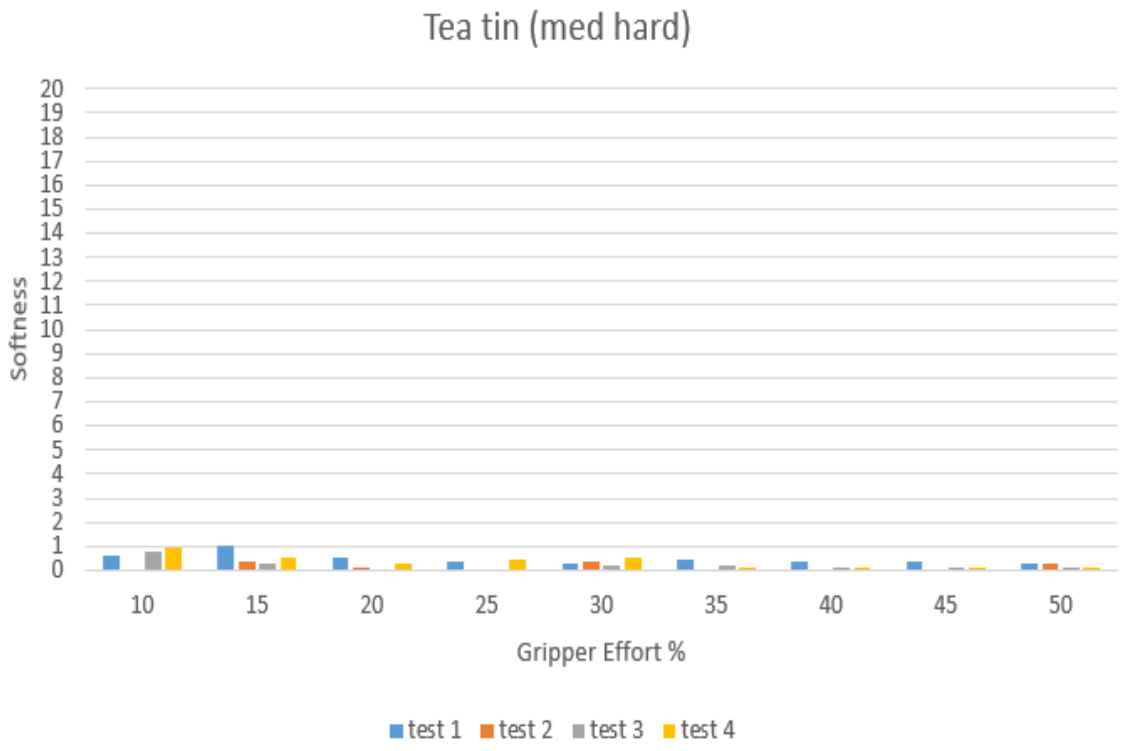
Figure 4.1: Gripper ready for small bear grasp

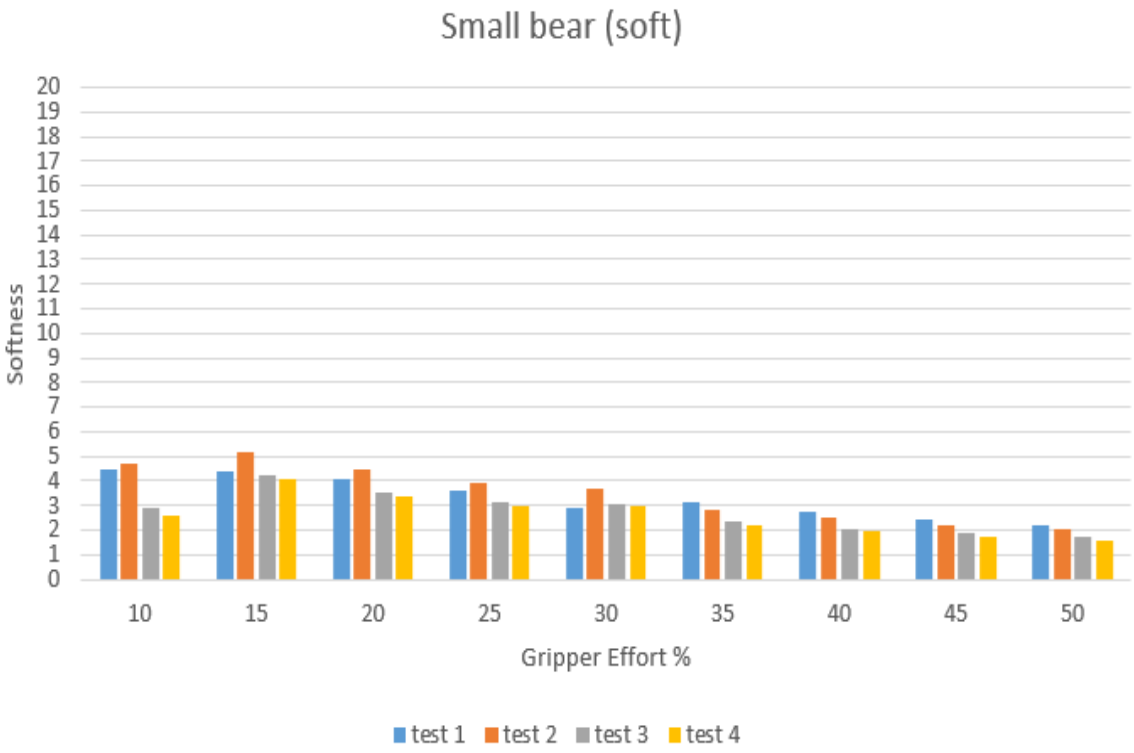
With $f = 5\%$ the gripper stops moving when it comes into contact with the surface of any of our objects. For this reason we use this for our initial grasp of each object. The servo position p_{start} is then saved to use when calculating the softness value after applying each additional force 10, 15, 20...50%.

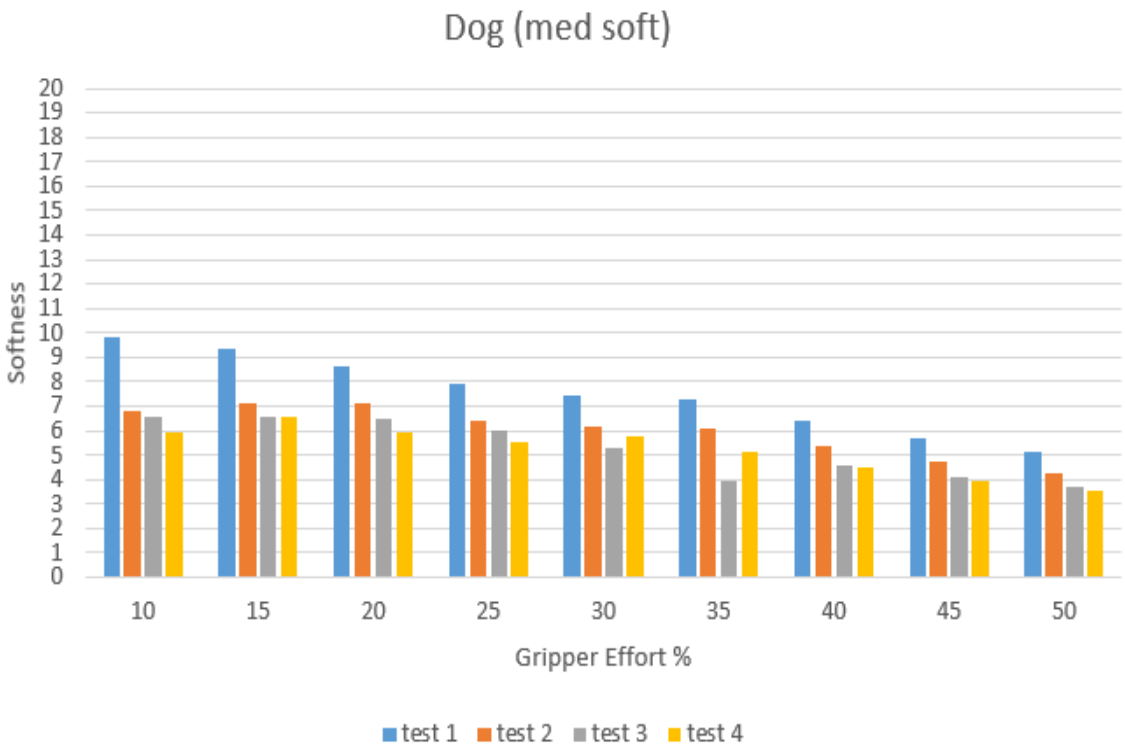
We ran our experiment on each object four times, the results are shown in the following bar charts:

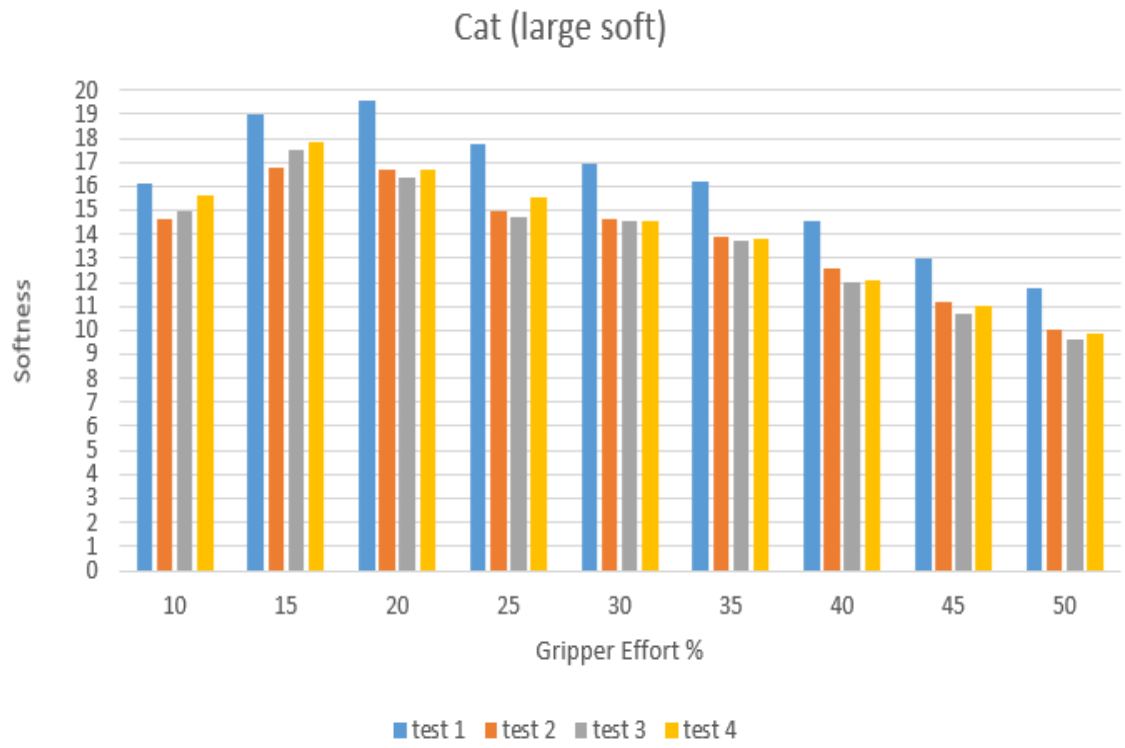





Tea tin (med hard)
Softness
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
10
15
20
25
30
35
40
45
50
Gripper Effort %
test 1
test 2
test 3
test 4


Small bear (soft)
Softness
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
10
15
20
25
30
35
40
45
50
Gripper Effort %
test 1
test 2
test 3
test 4


Dog (med soft)
Softness
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
10
15
20
25
30
35
40
45
50
Gripper Effort %
test 1
test 2
test 3
test 4


Cat (large soft)
Softness
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
10
15
20
25
30
35
40
45
50
Gripper Effort %
test 1
test 2
test 3
test 4

Result

We have found there is a significant difference between the softness values for objects in H and objects in S. This is particularly evident when applying forces at the lower end of our scale. As expected the H set had softness values close to zero for all applied forces. For the S set we found that softness increased up until f reached 15% or 20% before decreasing as f increases further. We have also found that there can be a significant difference the softness measured at a specific force over a number of grasps. This could be down to objects moving slightly between grasps causing the gripper to grasp in a different position.

4.1.2 Experiment 2

Overview

In our previous experiment an objects softness value was typically at its highest when $f = 15$. As our intention is to build a system which will allow the robot to detect a softness value by applying a minimal amount of forces, we decided to only detect softness at force $f = 15$ for our next experiment. This will also allow us to evaluate how much variation there is over a higher number of grasps at the same force.

The results also show that the objects $\in H$ gave softness readings at or bellow 1.3 when applying this force. Thus, we expect any objects with softness bellow or equal this value in our next experiment to be members of this set. However, we cannot say any with values above should be $\in S$ as we intend to add objects to H which might have readings slightly above any of the original set.

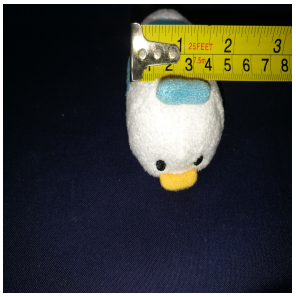
This experiment was designed to find if the softness values calculated at $f = 15$ during the previous experiment were repeatable and whether we would find similar results when detecting softness at this force for a new set of objects.

Test Set

We introduced eight items into this experiment, four which we classified to belong to each set S and H. One item classified to each set was predicted to have a softness value which would be closer to items in the other set than any in the previous experiment: one small soft toy duck measuring 40mm in width and one Pop Tarts box measuring 120mm, which although fairly rigid could easily be crushed by a human hand.

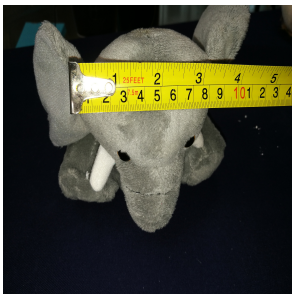
The objects added to the two sets of objects from the previous experiment are detailed bellow:

Soft Set Additions



Child's soft duck toy, measuring around 40mm at grasp point

Child's soft monkey toy, measuring around 90mm at grasp point



Child's soft elephant toy, measuring around 90mm at grasp point

Child's soft bear toy, measuring around 110mm at grasp point

Hard Set Additions

Oxo cube box, measuring around 60mm at grasp point

Pop tarts box, measuring around 90mm at grasp point

Hot-chocolate container, measuring around 75mm at grasp point

Coffee container measuring around 90mm at grasp point

Design

For each object, 10 grasps were applied to calculate a softness value at $f = 15$ by using a similar method to the previous experiment, differing only in that the softness value was only calculated at $f = 15$ and that 10 different grasps were applied to each object to obtain 10 softness readings. The results of this experiment are shown in the bellow graph:

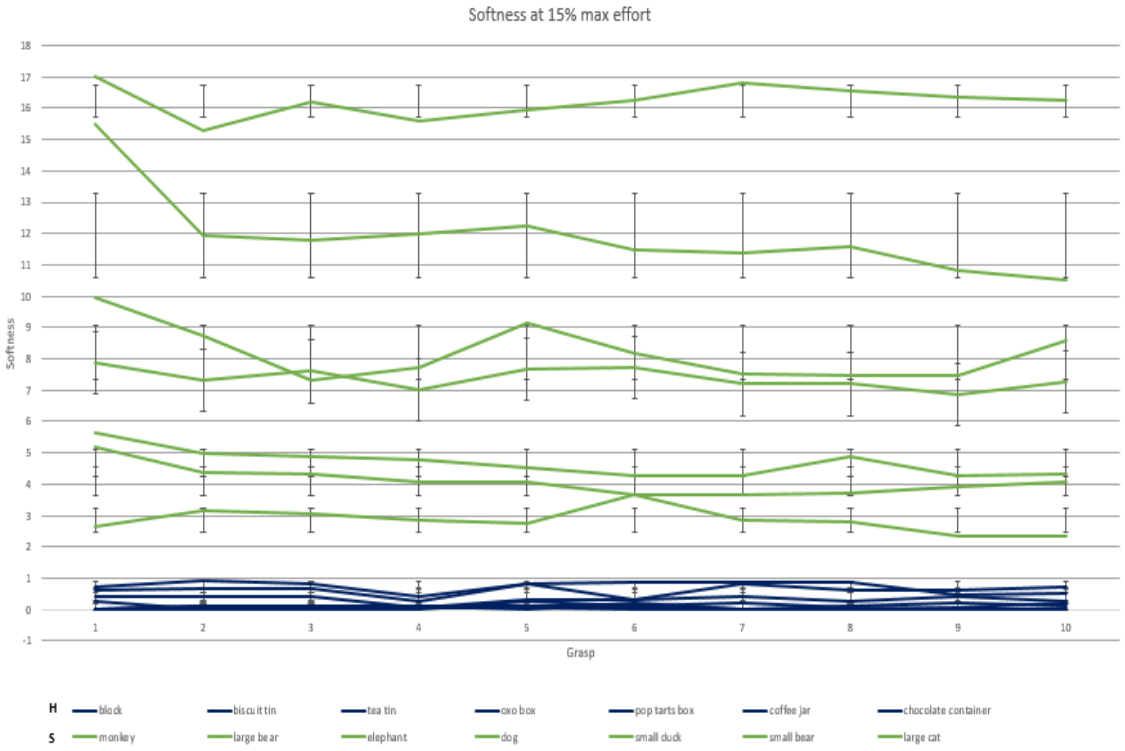


Figure 4.2: Softness Detected at Max Effort 15.

Result

Each item from the training set which was classified to be $\in H$ had softness values for each grasp bellow 1; the newly added objects which we hypothesised to be in this class were also all bellow 1. This was a unexpected as the maximum softness for any object in this set in the previous experiment was 1.3 at this force. Additionally we added the Pop Tarts box to the set, which was expected to be the 'softest' item $\in H$.

Each object which we assigned to S returned readings at or above 2.3 (which is the lowest softness value detected across both experiments for items in S). This value was detected when grasping the small soft duck toy which was predicted to be close to the limits of what the gripper could detect to be 'soft'. The results, therefore, show there was a clear distinction between the two sets of objects which we can use to classify as 'hard' or 'soft'.

4.1.3 Experiment 3

Overview

As the $softness\_bubble\_sort()$ method sorts objects into an increasing order of softness, we required these objects to be ordered by a human perception of softness to enable us to evaluate it's correctness. We therefore carried out a small experiment where seven participants where asked to order a set of objects in order of decreasing softness from left to right.

Test Set

Our test set for this experiment included three items form our previous experiment: the oxo box, small bear and wooden block. We also added a new object to this experiment as we were limited to the size of objects we could use due to the maximum opening size of Baxter's left gripper which is around 100mm.

Figure 4.3: Snowman object

Design

Each participant was given a canvas bag containing the objects and asked to place them in the desired order of softness on a desk. The conductor then stood back to give the participant space while observing the experiment. The participants were given no time limit to complete this experiment.

Result

The results from this small experiment gave us a high confidence of the order the objects should be placed in as all but one participant placed the objects in the same order: small bear, snowman, oxo box, wooden block. The other participant though placed the two middle objects in the opposite order. We should also note that two of the other participants took seemed to a little unsure of what order to place these two objects in before arriving at their final decision.

Nonetheless, as over 85% of our participants ordered the objects by decreasing softness: small bear, snowman, oxo box, wooden block; we used this as the correct order which we should expect Baxter to place the objects in after $softness\_bubble\_sort()$ is called.

We have included the full results and ethics forms for this experiment in Appendix B

4.1.4 Experiment 4

Overview

This purpose of this experiment was to evaluate our implementation of our $softness\_bubble\_sort()$ method against the results of the previous experiment.

Before the experiment we placed two objects on the table and ran the program to make sure the grippers would move to the correct locations to grasp them. We found the both where $\approx +20mm$ out on each axis when trying to grasp the objects. We therefore, subtracted $20mm$ on each axis from the poses sent to the inverse kinematics service within the $goto\_position()$ method e.g.

```
'left': PoseStamped(
    header=hdr,
    pose=Pose(
        position=Point(
            x=pos_x - 0.02,
            y=pos_y - 0.02,
            z=pos_z - 0.02,
        ),
        orientation=Quaternion(
            x=q_x,
            y=q_y,
            z=q_z,
        ),
    ),
),
```

Figure 4.4: Left Pose with 20mm offset on x, y and z axes

We then found the grippers would go to the correct locations to grasp objects using the retrieved coordinates from the $detect\_pcl\_objects$ service.

Test Set

Our test set for this experiment was the same set as used in the previous experiment 4.1.3

Design

The objects were placed in-front of Baxter in three different orders before calling the sorting function:

- oxo box, wooden block, small bear, snowman
- small bear, oxo box, wooden block, snowman
- wooden block, snowman, oxo box, small bear

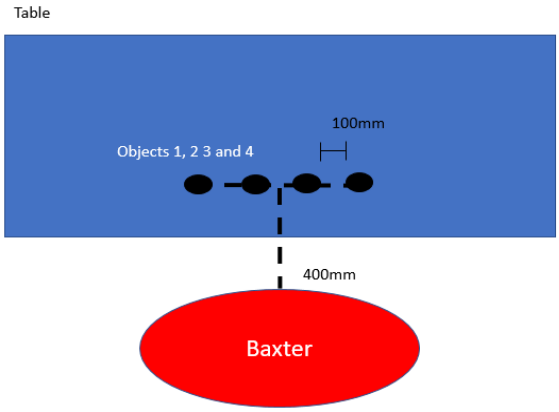


Figure 4.5: Approximate positions of objects during experiment

The observer then watched the robot to evaluate how/if the sorting task was completed and the experiment was videoed to allow evaluation further at a later point.

Result

Each time the function was executed Baxter sorted the objects into the order we concluded to be correct from the previous experiment: small bear, snowman, oxo box, wooden block. We should note however, that during the initial run, when placing the snowman, Baxter knocked it over after it was placed in its position.

4.1.5 Experiment 5

Overview

This purpose of this experiment was to evaluate our implementation of our $move\_soft\_objects\_to\_bin()$ method. We expect Baxter to remove any 'soft' items from the table while leaving any which are perceived to be 'hard'. The function was called three times, each with a different arrangement and set of objects.

Like the previous experiment we placed two objects on the table before the experiment and ran the program to make sure the right gripper would move to the correct location to grasp them. We found this time coordinates required an offset of $\approx +80mm$ to be added to the x-axis while the y-axis required an offset of $\approx -20mm$. The z-axis did not require an offset in this case.

Test Set

The following objects from the sets used in experiments 1: 4.1.1 and 2: 4.1.2, were used for this experiment.

- 1st execution of $move\_soft\_objects\_to\_bin()$:
  1. Wooden block from set H
  2. Soft Dog toy from set S

3. Small bear from set S

- 2nd execution of $move\_soft\_objects\_to\_bin()$:
  1. Wooden block from set H
  2. Dog toy from set S
  3. Tea container from set S
- 3rd execution of $move\_soft\_objects\_to\_bin()$:
  1. Small duck toy from set S
  2. Elephant from set S
  3. Hot-chocolate container from set H

Design

Three objects were placed in-front of Baxter, in a triangular fashion so the middle object's centroid was in a position which was $\approx +$ or $-$ $200mm$ different to the other two objects on the x axis. We conducted this experiment three times, placing the second object $\approx +200mm$ on the x axis in the first and third executions and $\approx -200mm$ for the second. The bin for the soft items to be dropped into was placed at the end of the table on Baxter's right-hand side.

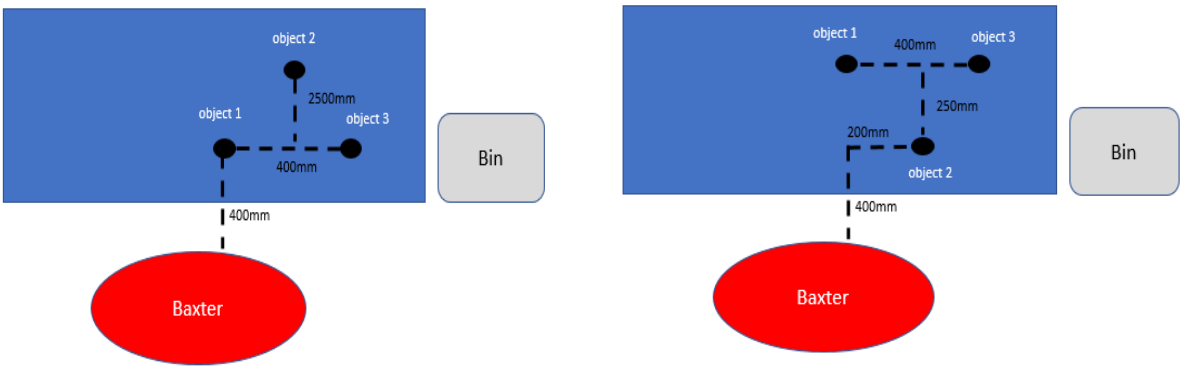


Figure 4.6: Approximate positions of objects during experiment

The position of the bin was added to the $ABOVE\_BIN\_POS$ variable in the $SoftnesSorter$ class statically by moving the gripper to the desired position to drop items into the bin and calling the $baxter\_interface.Limb('right')$ $.get\_joint\_angles()$ method.

The observer then watched the robot to evaluate how/if the sorting task was completed and the experiment was videoed to allow evaluation further at a later point.

Result

Each time the function was executed Baxter removed the correct items and dropped them in the bin. While watching the recorded video however, we noticed the yaw must have been out in the point cloud data gained

using the static transform as the robot would grasp objects on the left side of the table with a noticeably greater percent of the gripper than it used for items on the right. The task was still completed by Baxter though as there was not enough error to cause the gripper to either miss an object on the right side or press against the table on the left.

Chapter 5

Conclusion

In this chapter we summarise our project and achievements before discussing future improvements to our system.

5.1 Summary of Project

We have achieved the following during the course of this project:

- Presented and implemented a method of measuring the softness of objects using Hooke's Law and the feedback from the EZgripper Servo.
- Segmented and filtered point clouds to successfully and efficiently retrieve object locations and dimensions.
- Used Baxter the Research Robot to demonstrate, test and evaluate the usefulness and reliability of this approach.

The original purpose of this project was to investigate the possible use of the EZGripper to acquire a measure of softness for objects. We have shown this to be possible and have built a system with ROS using three packages which allow Baxter to demonstrate this functionality by sorting items by their perceived softness.

Of these three packages we have altered the existing $ezgripper\_driver$ package to add a service which returns the current servo position, and created two new packages:

- The $glasgow\_cluster\_detection$ package which allows objects locations and dimensions to be found efficiently using a Kinect sensor and 3D-point-clouds.
- The $glasgow\_softness\_detection$ package which utilises the services made available by the $glasgow\_cluster\_detection$ and the $ezgripper\_driver$ packages, to enable Baxter the robot to locate and sort objects by a measure of softness calculated using Hooke's Law.

We have achieved all of the high priority requirements which were needed to make this project successfully, we have tested our system over a number of experiments showing that it allows Baxter to successfully order objects by softness or separate them into 'hard' and 'soft' sets.

In the requirements for this project we stated if we had sufficient time we hoped to add some object recognition capabilities to the system. Unfortunately, this has not been possible due to the work load and time constraints of the project.

Instructions for installing and running the software we have developed through this project can be found in appendix A

5.2 Future Work

5.2.1 Continuous Object Detection

Our $cluster\_detection$ node is capable of returning object location and dimension information any-time its service is called. We could utilise this by calling the service to get new new information any-time Baxter picks or places an object. Which would allow us to detect if a new object has been added to the scene or if Baxter has knocked one over when placing. The reason we did not add this functionality was mainly due to time constraints and the possibility that Baxter's limbs may be detected as objects while they are in the process of picking or placing. This problem can be avoided though by using MoveIt [7] to publish a point cloud topic with which has the robot removed from the scene.

5.2.2 object recognition

PCL includes a lot of functionality for object recognition using point clouds. This would be useful as we could store information about the softness of objects to allow the robot to use softness as a means of gaining a higher confidence when recognising objects, and predicting the softness of objects using recognition.

5.2.3 Using Softness Measure to Determine Force

As hard objects are often more dense and heavier than soft objects the system could be extended to apply additional force depending on the perceived softness measure. This could allow the robot to pick up objects such as paper cups without squashing them, while also allowing heavier objects to be grasped without dropping them due to insufficient applied force.

5.2.4 Object Rotation Detection

We have made no effort due to the time constraints of the project, to detect the rotation of objects to allow Baxter to rotate the gripers in order to grasp them at a good angle. We did not think this was worthwhile functionality to spend time on as we were always mindful of how the gripper would grasp each object when we were placing them. We have noticed however, during experiments, when objects are placed by Baxter they will occasional be at a different rotation to when they picked up. We have not had any problems with this thus far, however, this his could be problematic in future executions of our sorting function as the left gripper is very limited to the width of objects it can grasp. This functionality would therefore be most useful if it is added after Continuous Object Detection so the rotation of object could be detected each time they are placed.

5.2.5 Additional EZGripper

The use of only one EZGripper limited the behaviour we could implement during our sorting functions:

- For $softness\_bubble\_sort$ this limited us to items which were upto $\approx 60mm$ wide due to the left gripper's much smaller grasp width. It also forced Baxter to grasp the first two objects with the EZGripper before allowing the left gripper to pick up the first, as we required a softness reading for both before they could be compared. We also feel we could have used a more impressive sorting algorithm using more objects if we could have detected softness with both grippers.
- For $move\_soft\_objects\_to\_bin()$ this limited the number of objects we could use as we could only place them on side of the table.

By adding another EZGripper we feel we could therefore, improve our sorting functionality to make it more efficient.

5.2.6 Fixed Kinect Location

Throughout the course of the project we have found that we have been required to re-calibrate the Kinect sensor multiple times due to both the Kinect and Baxter being in unfixed positions while other students need to work in the same space, on similar projects. To avoid this problem a fixed position for the Kinect is required.

5.2.7 More Robust Gripper Tendon

We have noticed during the project that the part of the tendon which connects to the servo appeared to have stretched, loosing some of its elasticity. This caused us to modify how we designed our sorting algorithms as commands which set the gripper to go to a position which is not fully opened or closed, no longer worked. Hence we could, not close the gripper by a certain percent to get ready to grasp an object, which limited the way in which we could position and grasp objects. There is therefore, as a need for a more robust tendon. Sake eclectronics have tried to address this issue with the EZGripper v2 [12] however, we would require extensive further testing to be sure they have resolved the problem.

Appendices

Appendix A

Instillation and Running the Software

These packages provide functionality to allow Baxter the robot to locate and sort objects by a measure of softness.

A.1 prerequisites

- Have a Baxter Research Robot from Rethink Electronic and have completed the workstation setup instructions at:
  http://sdk.rethinkrobotics.com/wiki/Workstation_Setup
- Have installed the iai_kinect2 ROS package by following the steps at:
  https://github.com/code-iai/iai_kinect2/blob/master/README.md
- Have an EZGripper from Sake electronics installed as Baxter's right gripper:
  http://sakerobotics.com/products2/
- Have a package for finding the pose of the kinect installed. We use: calibration_glasgow [6]
  https://github.com/gerac83/glasgow_calibration. Once the pose is found it should be added to the static transform in the softness_sorting launch file.

A.2 Instillation

These packages can be installed using the following steps:

- Each package should be placed in the src directory within your ROS/catkin workspace
- Build workspace using the following command form the root directory of the workspace: $catkin\_make$

A.3 Running

To run the softness sorting function, first start roscore then from a new terminal use:

- roslaunch glasgow_softness_detection softness_sorting.launch -sort

or to run remove_soft_objects function use:

- roslaunch glasgow_softness_detection softness_sorting.launch -remove

or to recalibrate the gripper tendon:

- roslaunch glasgow_softness_detection softness_sorting.launch -calibrate

A.4 Cluster Detection

The cluster detection node can also be run by its own if you wish to use the detect_cluster_location service with another ROS node. This can be done using the following command:

- roslaunch glasgow_object_detection cluster_detection.launch

Appendix B

Experiment 3 Results and Ethics Forms

B.1 Results

Softness Ranking experiment 09/03/2016

Ranking depending on softness with hardest object at rank 4 up to softest at rank 1

Participant	1	2	3	4
1	bear	snowman	Oxo box	Wooden block
2	Bear	snowman	Oxo box	Wooden block
3	Bear	snowman	Oxo box	Wooden block
4	Bear	Oxo box	Snowman	Wooden bloak
5	Bear	snowman	Oxo box	Wooden block
6	Bear	snowman	Oxo box	Wooden block
7	Bear	snowman	Oxo box	Wooden block

B.2 Intro

Softness Ranking Investigation,

2016/17 Craig Hamill

The aim of this experiment is to investigate how humans will rank these four items depending on softness. We have developed a software package which allows Baxter the robot to detect a softness measure for objects before sorting them based on this value. To evaluate its correctness, we must first have a set of objects for which we are confident of how they would be ranked by humans. I will give you a bag of four objects, then will ask you to take each from the bag and place them on the desk in order of softness. Place the object you perceive to be the softest at the rightmost position stepping left as they decrease in softness.

I will be observing you while you perform the tasks and will take note of the order you place the items in at the end of the experiment Please ask questions if you need to and please let me know when you are finished.

I may ask you some questions at the end of the experiment. Please remember that it is the objects, not you, that is being evaluated You are welcome to withdraw from the experiment at any time Do you agree to taking part in this evaluation? Do you have any questions before we start?

B.3 Ethics Checklist Form

School of Computing Science
University of Glasgow

Ethics checklist form for assessed exercises (at all levels)

This form is only applicable for assessed exercises that use other people ('participants') for the collection of information, typically in getting comments about a system or a system design, or getting information about how a system could be used, or evaluating a working system.

If no other people have been involved in the collection of information, then you do not need to complete this form.

If your evaluation does not comply with any one or more of the points below, please contact the Department Ethics Committee for advice.

If your evaluation does comply with all the points below, please sign this form and submit it with your assessed work.

---

1. Participants were not exposed to any risks greater than those encountered in their normal working life.

   Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that occur outside usual laboratory areas, or that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, that use sensory deprivation (e.g. ear plugs or blindfolds), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback

2. The experimental materials were paper-based, or comprised software running on standard hardware.

   Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, mobile phones, and PDAs is considered non-standard.

3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.

   If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, or the data is to be published), then signed consent is necessary. A separate consent form should be signed by each participant.

   Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script.

4. No incentives were offered to the participants.

   The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

5. No information about the evaluation or materials was intentionally withheld from the participants.
   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

6. No participant was under the age of 16.
   Parental consent is required for participants under the age of 16.

7. No participant has an impairment that may limit their understanding or communication.
   Additional consent is required for participants with impairments.

8. Neither I nor my supervisor is in a position of authority or influence over any of the participants.
   A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. All participants were informed that they could withdraw at any time.
   All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script.

10. All participants have been informed of my contact details.
    All participants must be able to contact the investigator after the investigation. They should be given the details of both student and module co-ordinator or supervisor as part of the debriefing.

11. The evaluation was discussed with all the participants at the end of the session, and all participants had the opportunity to ask questions.
    The student must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation.

12. *All the data collected from the participants is stored in an anonymous form.*
    All participant data (hard-copy and soft-copy) should be stored securely, and in anonymous form.

Module and Assessment Name Level 4 Project

Student's Name Craig Hamill

Student's Registration Number 2094131

Student's Signature C.Hamill

Date 09/03/2017

Figure B.1: Ethics Checklist

B.4 Debriefing

Debriefing

The aim of this experiment was to investigate how humans will rank these four items depending on softness. We have developed a software package which allows Baxter the robot to detect a softness measure for objects before sorting them based on this value. To evaluate its correctness, we must first have a set of objects for which we are confident of how they would be ranked by humans. We therefore, asked you to take part in this experiment to help us gather enough data to evaluate our software. I gave you a bag of four objects, then asked you to take each from the bag and place them on the desk in order of softness.

Do you have any comments or questions about the experiment? Please take a note of my email address (2094131h@student.gla.ac.uk) and the email address of my project c co-ordinator (Paul.Siebert@glasgow.ac.uk), and please let us know if you have any further questions about this experiment Thank you for your help [thank the participant].

Bibliography

[1] Wolfgang FJ Deeg. *The analysis of dislocation, crack, and inclusion problems in piezoelectric solids*. PhD thesis, Stanford University, CA, 1980.

[2] Willow Garage. Robots/pr2 - ros wiki.

[3] Erico Guizzo and Evan Ackerman. Dynamixel mx-64t - robotis. lhttp://www.robotis.us/dynamixel-mx-64t/. (Accessed on 03/13/2017).

[4] Erico Guizzo and Evan Ackerman. How rethink robotics built its new baxter robot worker, 2012.

[5] Oliver Kerpa, Karsten Weiss, and Heinz Worn. Development of a flexible tactile sensor system for a humanoid robot. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 1–6. IEEE, 2003.

[6] Aamir Khan, Gerardo Aragon-Camarasa, Li Sun, and J Paul Siebert. On the calibration of active binocular and rgbd vision systems for dual-arm robots. 2016.

[7] MoveIt. 3d perception/configuration tutorial moveit_tutorials indigo documentation. http://docs.ros.org/indigo/api/moveit_tutorials/html/doc/pr2_tutorials/planning/src/doc/perception_configuration.html?h (Accessed on 03/19/2017).

[8] Sadao Omata, Yoshinobu Murayama, and Christos E Constantinou. Real time robotic tactile sensor system for the determination of the physical properties of biomaterials. *Sensors and Actuators A: Physical*, 112(2):278–285, 2004.

[9] Sadao Omata and Yoshikazu Terunuma. New tactile sensor like the human hand and its applications. *Sensors and Actuators A: Physical*, 35(1):9–15, 1992.

[10] PCL. Documentation - point cloud library (pcl). http://www.pointclouds.org/documentation/tutorials/cluster_extraction.p (Accessed on 03/14/2017).

[11] Rethink Robotics. Baxter - rethink robotics.

[12] Sake Robotics. ezgripper robot grippers hands — sake robotics. lhttp://sakerobotics.com/products2/. (Accessed on 03/19/2017).

[13] Sake Robotics. Robot grippers — sake robotics. http://sakerobotics.com.

[14] ROS. Ros.org — powering the world's robots. http://www.ros.org/.

[15] J Rychlewski. On hooke's law. *Journal of Applied Mathematics and Mechanics*, 48(3):303–314, 1984.

[16] Jae S Son, Eduardo A Monteverde, and Robert D Howe. A tactile sensor for localizing transient events in manipulation. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 471–476. IEEE, 1994.

[17] MW Strohmayr, Hannes P Saal, AH Potdar, and P Van Der Smagt. The dlr touch sensor i: A flexible tactile sensor for robotic hands based on a crossed-wire approach. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 897–903. IEEE, 2010.

[18] Theimo Wiedemeyer. code-iai/iai_kinect2: Tools for using the kinect one (kinect v2) in ros. https://github.com/code-iai/iai_kinect2. (Accessed on 03/11/2017).