Carnegie Mellon University

16-681

MRSD Project I

Task 6 Progress Review 1
Team C - COBORG

Jonathan Lord-Fonda

Teammates: Husam Wadi, Feng Xiang, Yuqing Qin, Gerry D'ascoli

March 04, 2021

Table of Contents

Individual Progress 1

Challenges 2

Teamwork 3

Plans 4

Appendices 5

Appendix 1 - main_state_machine Node Code 5

1. Individual Progress

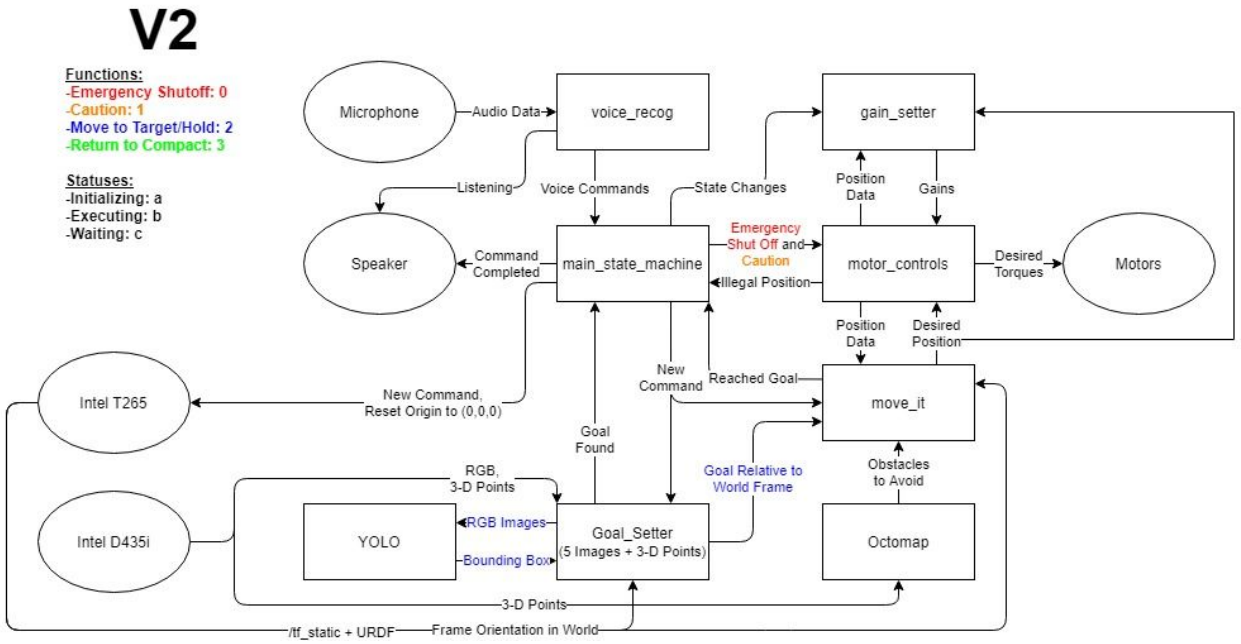
My primary task since the previous Individual Lab Report (ILR) was to update the ROS Node Maps, as seen in Figure 1 and Figure 2. I added the functionality of a gain_setter node, which will allow the system to adjust its control characteristics as it executes different tasks and parts of those tasks. For example, the gain_setter node may set a high proportional gain for the motors at the beginning of the robot's trajectory so that it can quickly approach its target and then lower the proportional gain and increase the derivative gain as it nears the target to prevent hard impact at close proximity. Once the arm is within a certain distance, the gain_setter node may switch control schemes entirely to an impedance-driven control process to promote better movement compensation of the arm.

The next update to the ROS Node Map involved altering the vision system. Initially the plan was to use a Localizer node to continuously generate transforms between the tracking camera and the global frame. After digging into the operation of the tracking camera, Husam noticed that the camera performed these transformations automatically and published them to /tf_static. With no more need to generate transformations in a node, the other operations encompassed by the Localizer node were wrapped into the Goal_Setter node and the Localizer node was eliminated.

The final change to the ROS Node Map was adding a speaker output. After setting up the voice_recog node, Gerry realized that auditory feedback to the user would help them understand when the COBORG was listening for a command and when it had successfully parsed one. I realized that it would also benefit the user to know when the COBORG system was done executing

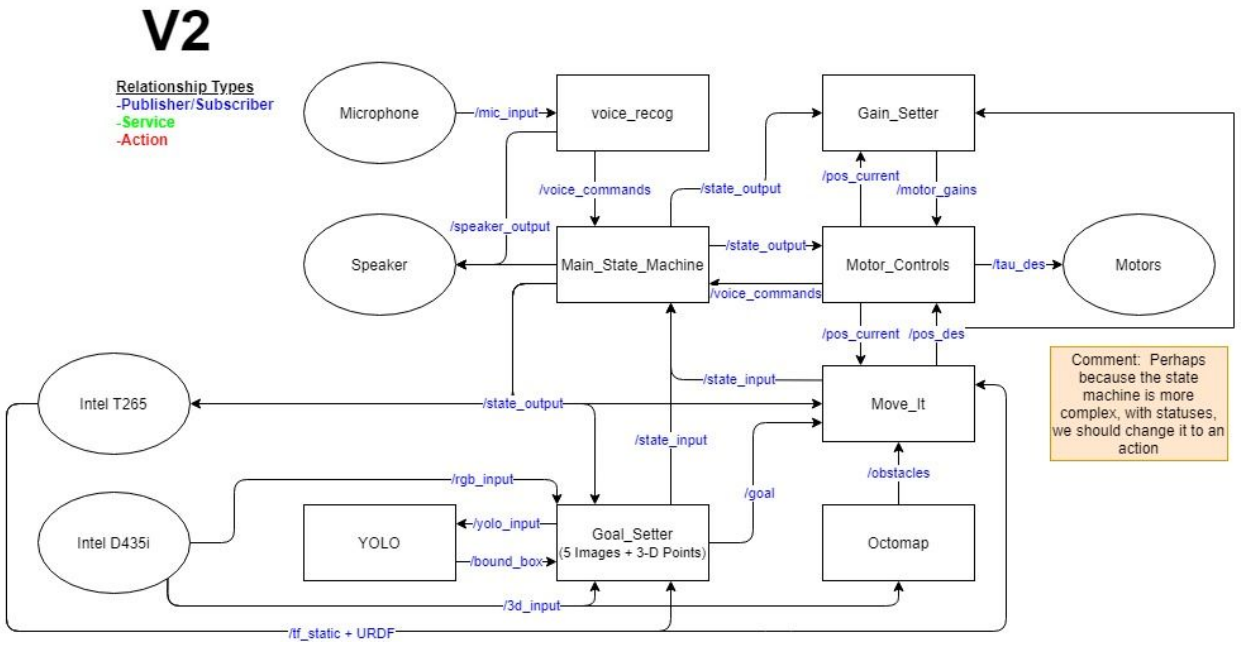
a command and ready for another; it would be particularly useful for the “Hold” command as it would let the user know that the COBORG was fully supporting the part and they could remove their hands. To that end I added the node with a topic flowing from both voice_recog and main_state_machine.

Figure 1 - Updated ROS Node Map, Semantic Descriptions



The primary changes that occurred between this version and the previous include removing the Localizer node and wrapping some of its functionality into the Goal_Setter node, as well as adding and connecting the Speaker and gain_setter nodes. Additional changes include small topic adjustments, such as sending the overall state to more nodes and adding a caution state on top of the emergency branch.

Figure 2 - Updated ROS Node Map, Node and Topic Descriptions



As described in the caption for Figure 1, a number of nodes and topics were added and named. Additionally, which is unique to this graph, the specific topics were rearranged to accommodate the increased connections; currently five separate nodes read /state_output, whereas only two were subscribed to this topic previously.

Besides working on the ROS Node Map I also coded out the main_state_machine node. Previously I had written a semantic description of how it would function, but this time I coded the entire system into ROS and Python. The full code for the node can be found in Appendix 1. Additionally, I wrote out a semantic description of the Goal_Setter node to facilitate discussion between Yuqing and I. It allowed me to accurately express my ideas to her so that she could review them, understand them, and highlight any problems that stood out to her. The full semantic description for the Goal_Setter node can be found in Appendix 2.

Alongside these other tasks I also helped Jason and Husam construct a structure to hold our COBORG robot and allow for full demonstrations. While rather simple, its modularity allows us to quickly adjust the height and angle of the board so that we can test our robot under varying circumstances and judge its performance accordingly.

2. Challenges

My primary goal listed for this review cycle was to implement the main_state_machine node in ROS and have it interface with the voice_recog

node, demonstrating that it could recognize and respond to commands from the voice system. Embedded in this task was the assumption that I would be able to finish installing Linux and ROS on a flash drive, write and test the code on my personal laptop, and then push the code to GitHub and use that to interface it with the voice system. Unfortunately, after spending roughly ten hours on installing Linux and ROS semi-successfully in previous weeks, I returned to my flash drive to find out that it did not retain any of the programs I had downloaded on my previous attempts. Apparently a full installation of Ubuntu on a flash drive is not the same thing as a persistent installation of Ubuntu on a flash drive. After another four or more hours of researching and installing I finally succeeded, but that was at 2:00 on the morning of the progress review, so I was unable to fully implement and integrate the main_state_machine node.

Deleting the Localizer node increased the complexity of the Goal_Setter node which now contains a number of different features. This increased complexity led me to discuss it with Yuqing at length and write up a semantic description of the node as I envisioned it, a task that I had not planned for. The effort required by this task was time well spent, but that was time unaccounted for in the plan that could have gone to one of the other challenges in this section. Additionally, the current semantic description doesn't go into detail and we're still unsure about how exactly we will transform the RGB images, 3-d point clouds, and pose frames to align so that we can compare them consistently.

Another challenge that cropped up while looking at the ROS Node Map was the effectiveness of our actuated manipulation branch. Currently we'd like to account for the user's movement while the robot is in motion, but with RRT Connect taking 0.5 to 1 seconds to generate a path, it is likely that we'll need to use other features to correct for the user's motion outside of our path planning. One possible feature we could implement would be a node that would continuously shift the most recent path's waypoints by small amounts to account for the user's movement. We could also continuously run RRT Connect for newer paths and dedicate our computer's GPU to RRT Connect paths to reduce the time taken to generate a path.

3. Teamwork

Over this past week, Jason created the URDF model for the COBORG system and integrated it with Move-It. In a great success for our project he was also able to assign the arm to move to a point selected in RVIZ and have it actually

move to the point. He also helped clean up our lab space and create a demonstration structure to hold the COBORG system.

Since the last progress review, Gerry redesigned the voice recognition system to understand “Coborg” as a trigger word while also retaining a full dictionary so that it could tell “Coborg” apart from other words. In addition to this he wrote a script that could translate verbal commands into instructions for the robot and demonstrated its functionality by integrating his script with an audio feedback system which he programmed with a variety of sounds.

Over the past week, Yuqing installed a ROS package for the Realsense D435i camera and launched YOLO v3 with hand detection. In order to bring these two features together, Yuqing created a ROS wrapper for the YOLO model and tested it with the D435i camera. In order to improve its speed she set it up to run on her computer’s GPU and achieved speeds around 200 frames per second, far more than is required for our project.

Since the last progress review, Husam cleaned up our lab space and created a structure to both hold the COBORG robot and demonstrate the system. He also adapted the timeline to account for the increased homework load in the team’s immediate future and performed some risk analysis accordingly. Husam also assisted Jason with understanding the Intel Realsense T265 and helped Yuqing with the Intel Realsense D435i. He also helped me by making some changes to the ROS Node Map and explaining to me how the T265 fit into the node map. Husam also set up the Github repository and ensured that the team utilized it for version control.

4. Plans

Before the next progress review I would like to fully implement the main_state_machine node in ROS and integrate it with the voice_recog node. Additionally I would like to write out a semantic description of the Gain_Setter node and apply any necessary changes to the ROS Node Map as they become apparent.

Besides the overarching project planning, I would also like to work alongside Jason on the actuated manipulation branch of the project. The first step would be to sit down with him and absorb all of the information he’s learned in regard to it. After that I would like to plan out a detailed roadmap for developing the actuated manipulation branch and take at least one task upon myself to

complete before the next progress review. What this task is will be determined by Jason himself.

In addition to the directly-related project tasks, I would also like to do the CAD work for our PCB assignment. While not directly a part of our project, this assignment will form a basis for part of our project that we design later.

5. Appendices

5.1. Appendix 1 - main_state_machine Node Code

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

import rospy
from std_msgs.msg import Int32
from std_msgs.msg import Char

# Updates List:
# Consider changing the state machine to work on actions instead of publisher/subscriber
# Figure out how to publish sounds to speaker

# Functions:
# 0 = e_stop > Shut off power to motors
# 1 = caution > Motors stop moving, but maintain with lower torque threshold
# 2 = hold > Move to and hold plate
# 3 = compact > Return to home position

# Statuses:
# a = initializing > Command received, but not executing yet (e.g. detecting hands)
# b = executing > Command being executed (e.g. moving to target)
# c = waiting > Command completed/performing "holding" task, ready for next command
(maintaining position in 3d space)

# Speaker Sounds:
# 0 = start-up > System has launched
# 1 = listening > Keyword “COBORG” identified, awaiting full command
# 2 = understood > Voice command understood
# 3 = not understood > Voice command not understood
# 4 = emergency > Emergency command detected
# 5 = waiting > Task completed, available for next command

# Function for /voice_command
def new_command(new_command):
    if new_command == 0:
```

```
        function = 0 # e_stop
        status = a # initializing
        state_output_pub.publish(function)
    else if new_command == 1:
        if function != 0:
            function = 1 # caution
            status = a # initializing
            state_output_pub.publish(function)
    else if new_command == 2:
        if status == c:
            function = 2 # hold
            status = a # initializing
            state_output_pub.publish(function)
    else if new_command == 3:
        if status == c:
            function = 3 # compact
            status = a # initializing
            state_output_pub.publish(function)

# Function for /state_input
def status_update(new_status):
    if new_status == b:
        status = b # executing
    if new_status == c:
        status = c # waiting
        speaker_output_pub.publish(5)

# Initializations
rospy.init_node('Main_State_Machine')
function = 3 # compact
status = a # initializing
voice_commands_sub = rospy.Subscriber('voice_commands', Int32, new_command)
state_output_pub = rospy.Publisher('/state_output', Int32, queue_size=1)
speaker_output_pub = rospy.Publisher('/speaker_output', Int32, queue_size=1)
state_input_sub = rospy.Subscriber('state_input', Char, status_update)
state_output_pub.publish(function)
speaker_output_pub.publish(0)

rospy.spin() # Should this just be spin()?
```

5.2. Appendix 2 - Goal_Setter Node Semantic Draft

Goal_Setter Intuitive Draft

On Start-up{

Initialize publisher to /state_input
Initialize publisher to /goal
Initialize publisher to /yolo_input
Initialize subscriber to /state_output
Initialize subscriber to /rgb_input
Initialize subscriber to /3d_input
Initialize subscriber to /tf_static
Initialize subscriber to /bound_box
}

If receiving a 2 from state_output (COBORG Hold command){
////////////////////// This could be a first step, essentially, can we pull in a set of data?
Create rgb vector
Create 3d points vector
Create /tf_static vector
Create bound_box vector
Add the current rgb image to rgb vector
Add the current 3d points to 3d point vector
Add the current pose to /tf_static vector
Publish rgb to /yolo_input
Create index i to iterate through all vectors
//////////////////////
}

On receiving an image from YOLO{
////////////////////// This could be a second step, testing if we can evaluate YOLO's results well and tuning the system.
////////////////////// If YOLO ends up being really accurate here, we can throw out most of the complexity later
Check to see whether the bounding box's accuracy passes a threshold.

If it doesn't, either throw the results out or simply add another data
set to each vector and publish the rgb to /yolo_input.

If the bounding box (or boxes) achieve a certain threshold of accuracy or
intersection, run Goal Finding Program (below)
//////////////////////
}

Goal Finding Program (runs after good enough results are achieved){
Create bounding box midpoints vector (Maybe create a "left" and "right" one?)
Create Goal Midpoints vector
Create Goal Locations vector
Create Goal Poses vector

////////////////////// This could be a third step, can we transform the images to a global reference?

////////////////////// If we're doing this I guess we might have to grab the 3-d coordinates of the bounding boxes

Use /tf_static vector to transform bounding boxes into the global frame of reference

//////////////////////

////////////////////// This could be a fourth step

Find intersection of leftmost bounding boxes

Find midpoint of leftmost bounding boxes (H/2,W/2), add it to the bounding box midpoints vector

Find intersection of rightmost bounding boxes

Find midpoint of rightmost bounding boxes (H/2,W/2), add it to the bounding box midpoints vector

Find Goal Midpoint between bounding box midpoints, add it to the Goal Midpoints vector

//////////////////////

////////////////////// This could be a fifth step, potentially done at the same time as the third step?

Use /tf_static vector to transform 3-d point clouds into the global frame of reference

//////////////////////

////////////////////// This could be a sixth step, similar to the fourth step, but using multi-dimensional splines because the 3-d point cloud is sparse

Change 3-d point clouds to splines

Find the depth of each point corresponding to Goal Midpoint in each of the 3-d point clouds, add the 3-d point to Goal Locations vector

//////////////////////

////////////////////// This could be a seventh step, can we get the pose of the plate from the 3-d point cloud? Again, we can use the spline.

For each of the Goal Midpoints in each of the 3-d point clouds, check a small area around the Goal Midpoint to determine the surface pose, add it to Goal Poses vector

^-This could be as simple as + and - 1 pixel vertically and horizontally to determine the x and y components of change in depth

//////////////////////

////////////////////// This could be an eighth step, if everything else works, can we smush it together and publish our final answer?

Average Goal Locations

Average Goal Poses

Publish the averaged Goal Locations and averaged Goal Poses to /goal

Publish "b" to /state_input to let the Main_State_Machine node know that the command is done initializing and is now executing

//////////////////////

}