Progress Review 2

Individual lab report – 03 | October 23, 2015

TEAM DAEDALUS

Submitted by:

Mohak Bhardwaj

Team Members:
Dorothy Kirlew
Richa Varma
Shivam Gautam

Pranav Maheshwari

1. Introduction

Keeping the goals defined by the team for Progress Review 2, the following tasks were undertaken by them team across various subsystems:

- Setting up of environment for interfacing Kinect with ROS and preliminary testing.
- Interfacing of actuator control board with ROS.
- Integration of ROS and Bluetooth Serial interface.
- Locking down on mobile platform to be ordered.
- Placement of order for communication hardware.
- Conceptual design of power distribution board.

2. Individual Progress

The following responsibilities were taken up by me:

- Setting up of environment for interfacing Kinect with ROS and preliminary testing.
- Integration of ROS and Bluetooth Serial Interface.
- Conceptual design of power distribution board.
- Setting up version control for software.
- Primary Point of contact with sponsor.

MS KINECT INTERFACING

As per the team's goals, I worked along with Shivam on researching the appropriate libraries for interfacing MS Kinect with ROS, setting them up and visualizing data using RVIZ.

In order to get started with the Kinect, it was first required to understand the functionalities and limitations of the various open-source libraries available for interfacing MS Kinect. Also, one of the prime requirements for the libraries was ROS compatibility.

As per the literature survey, it was quite clear to me that we would have to use a library that provides functionality to manipulate point clouds with ease. Also, since the vision subsystem will involve highly complicated and computationally heavy algorithms, it is important for the libraries to provide easy-to-use abstractions with sufficient documentation and support. This is necessary to ensure that time is not wasted in understanding the internal working of the functions.

During the survey of libraries, I came across the following Kinect libraries that were ROS compliant:

- freenect_stack (libfreenect)
- openni_kinect
- kinect_aux

These libraries are open-source libraries dedicated to Kinect (openNI supports all natural interface sensors) and work well with ROS. Another library that was very appealing is the **Point Cloud Library (PCL)**, which is a general purpose library for manipulation point cloud data.

As per our literature survey on vision based obstacle avoidance algorithms, the paper that was finally narrowed down to was

[1] MS Thesis of Rasoul Mojtahedazdeh on Robot Obstacle Avoidance using MS Kinect –KTH Sweden

Where openNI has been used for Kinect interfacing. Also, the Point Cloud Library has massive amount of documentation and provides a large number of functionalities. Hence, it was decided that we would be leveraging the potential of both **OpenNI and PCL**.

Both Shivam and I set out to installing the required libraries in our machines. Fortunately, I found a tutorial that clearly mentioned the steps for downloading and installing the all required dependencies and core files for PCL and OpenNI. There were also steps for visualizing point clouds using OpenNIViewer. I installed all required libraries as well as auxiliary libraries such as CUDA for GPU processing which might be used in the future if the need arises. Although PCL and OpenNI should ideally come installed with ROS, but in some cases unmet dependencies need to be resolved as was the case in my machine. Therefore, just to be sure that there are no glitches, I compiled PCL from source. I also tested the libraries by visualizing the point cloud data obtained from the Kinect.

In addition to this, the point cloud data was also visualized on RVIZ using the openNI node in ROS. This verified that our environment is indeed functional and we can proceed further.

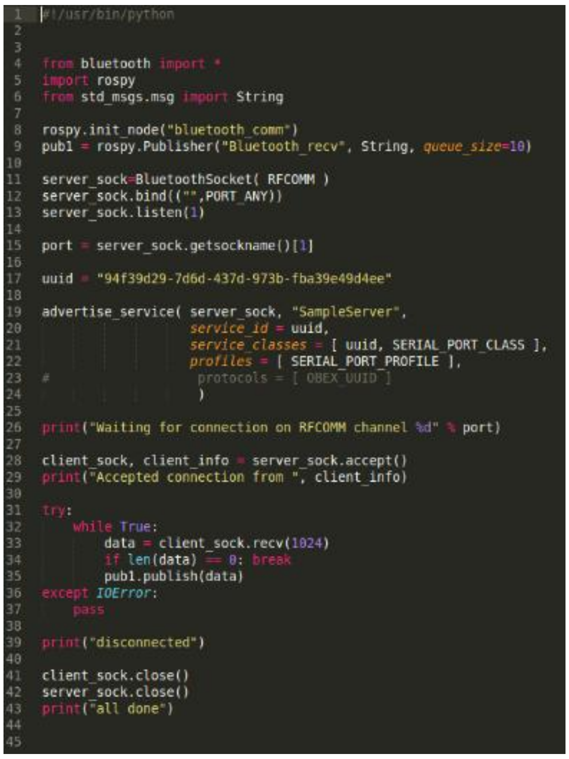
INTEGRATION OF ROS AND BLUETOOTH INTERFACE

Last week I had successfully created a bash script to listen on a RFCOMM port and receive commands sent by the phone serially over Bluetooth. Although, the interface was working properly, we realized that it would be better if a ROS node could be created for the same. The node would be constantly listening for serial commands over Bluetooth and publish on a topic whenever a new command is received.

In order to achieve this task I decided to read up on Bluetooth programming fundamentals. I spent some time learning about RFCOMM protocol, socket programming and Service Discovery Protocol (SDP). After understanding the basics, I worked with Pranav on configuring a ROS node using the python PyBlueZ library for Bluetooth serial communications. A server application for listening on a RFCOMM port with appropriate UUID for serial service was written within a ROS node framework. The node can now publish the ‘Park’ and ‘Return’ commands received serially via Bluetooth on a ROS topic (see Figure 1 for code snippet).

CONCEPTUAL DESIGN OF PCB

I also worked on the conceptual design of the PCB power distribution board for PCB Task. My specific role was going through the technical specifications of the major components namely, MS Kinect, Odroid XU4 and Arduino Mega and finding out the power requirements for each component. Also, I had to come up with the manual switch requirements for different components.



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


from bluetooth import *
import rospy
from std_msgs.msg import String

rospy.init_node("bluetooth_comm")
pub1 = rospy.Publisher("Bluetooth_recv", String, queue_size=10)

server_sock=BluetoothSocket( RFCOMM )
server_sock.bind(("",PORT_ANY))
server_sock.listen(1)

port = server_sock.getsockname()[1]

uuid = "94f39d29-7d6d-437d-973b-fba39e49d4ee"

advertise_service( server_sock, "SampleServer",
                   service_id = uuid,
                   service_classes = [ uuid, SERIAL_PORT_CLASS ],
                   profiles = [ SERIAL_PORT_PROFILE ],
#                   protocols = [ OBEX_UUID ]
                    )

print("Waiting for connection on RFCOMM channel %d" % port)

client_sock, client_info = server_sock.accept()
print("Accepted connection from ", client_info)

try:
    while True:
        data = client_sock.recv(1024)
        if len(data) == 0: break
        pub1.publish(data)
except IOError:
    pass

print("disconnected")

client_sock.close()
server_sock.close()
print("all done")
```

Figure 1: Code for Bluetooth listener ROS node

PRIMARY POINT OF CONTACT WITH SPONSOR

As my team's primary point of contact with the sponsor, I had to make sure that all communication with the sponsor go smoothly and there are no delays. Also, we are currently going through negotiations with the sponsor on ROS based platforms and arbitrating on that required me to keep the whole team in the loop at all times. Recently we have been facing issues with the sponsor not being responsive enough which is causing delays in our project schedule.

3. Challenges

The challenges that I had to face were:

4. Teamwork

The other members of my team were working on the following subsystems:

5. Plans

6. REFERENCES