

Checkpoint # 1 Report

1. Purpose:

The purpose of this checkpoint is to ensure that the development environment is properly configured. In this checkpoint, the initial step is to install the operating system using the official Ubuntu Mate image file. Next, we proceed to install ROS (Robot Operating System) within the environment. Afterward, we utilize the ROS roserial package, available on the official ROS Arduino website, to establish communication between the Raspberry Pi and Arduino.

2. Description of Design:

This checkpoint encompasses two key tasks: (1) Utilize the SSH command to establish a remote connection from your PC to the Raspberry Pi. Use the "rosversion" command to confirm the successful installation of ROS Melodic on the Raspberry Pi. (2) Employ commands and libraries from the roserial package to create a program featuring publishers and subscribers. This program facilitates bidirectional communication between the Raspberry Pi and Arduino. Specifically, the Raspberry Pi will transmit a numeric value to the Arduino, which will then multiply it by 2 and return the result to the Raspberry Pi.

In the described process, we have designed a system with publishers and subscribers, involving two topics. One topic is responsible for publishing numeric messages to the Arduino, while the other handles messages containing the multiplied values returned from the Arduino.

For the Raspberry Pi, we've developed a program that comprises a publisher responsible for sending messages under the topic "num" to the Arduino. Simultaneously, a subscriber on the Raspberry Pi subscribes to the "multiple" topic, which receives the multiplied numeric values from the Arduino.

In the Arduino's design, it also features a publisher and a subscriber. The Arduino subscribes to the "num" topic from the Raspberry Pi, retrieves the numeric value, performs the multiplication operation, and publishes the resulting value back to the Raspberry Pi under the "multiple" topic.

Following the construction of these programs within the ROS environment for the Raspberry Pi, we've created a launch file. This launch file enables the execution of multiple scripts in the terminal. Subsequently, we will delve into the implementation details.

Raspberry Pi:

(1) publisher – pi_publisher.cpp

```
#include "ros/ros.h"
#include "std_msgs/Int32.h"
#include <iostream>

class pi : public ros::NodeHandle
{
public:
    pi()
    {
```

```

        publisher = this->advertise<std_msgs::Int32>("num", 10);
        msg.data = 1;
    }

    void publishMessage()
    {
        publisher.publish(msg);
        std::cout<<"user's input is "<< msg.data <<std::endl;
    }

    std_msgs::Int32 msg;

private:
    ros::NodeHandle nh;
    ros::Publisher publisher;
};

int main(int argc, char **argv)
{
    ros::init(argc, argv, "pi_publisher");
    ros::NodeHandle nh;
    auto node = std::make_shared<pi>();
    while (ros::ok())
    {
        int input;
        // std::cout << "-----" << std::endl;
        // std::cout << "Enter user's input or enter -1 to quit: " << std::endl;
        std::cin >> input;
        if(input == -1) ros::shutdown();
        else node->msg.data = input;
        node->publishMessage();
    }

    ros::spin();
    return 0;
}

```

(2) subscriber – pi_subscriber.cpp

```

#include "ros/ros.h"
#include "std_msgs/Int32.h"
#include <iostream>

class pi : public ros::NodeHandle
{
public:
    pi()
    {
        std::cout << "Start to subscribe!" << std::endl;
        subscriber = this->subscribe<std_msgs::Int32>("multiple", 10,

```

```

        &pi::MessageCallback, this);
    }

void MessageCallback(const std_msgs::Int32::ConstPtr& msg_ptr)
{
    ros::Rate rate(1);
    msg.data = msg_ptr -> data;
    // std::cout<<"-----"<< std::endl;
    std::cout<<"message from Arduino is "<< msg.data <<std::endl;
    rate.sleep();
}

private:
// ros::NodeHandle nh;
ros::Subscriber subscriber;
std::string topic_name_multiple;
std_msgs::Int32 msg;
};

int main(int argc, char **argv)
{
    ros::init(argc, argv, "pi_sublsher");
    auto node = std::make_shared<pi>();
    ros::spin();
    return 0;
}

```

(3) launch file – amr.launch

```

<launch>
  <node name="pi_publisher" pkg="amr"
    type="pi_publisher" output="screen" />
  <node name="pi_sublsher" pkg="amr"
    type="pi_subscriber" output="screen" />
</launch>

```

Arduino:

(1) ino file – checkpoint1.ino

```

#include<ros.h>
#include<std_msgs/Int32.h>

ros::NodeHandle nh;
std_msgs::Int32 msg;

ros::Publisher publisher("multiple", &msg);
ros::Subscriber<std_msgs::Int32> subscriber("num", &messageCb);

void messageCb(const std_msgs::Int32& received_msg)

```

```

{
    msg.data = received_msg.data*2;
    publisher.publish(&msg);
}

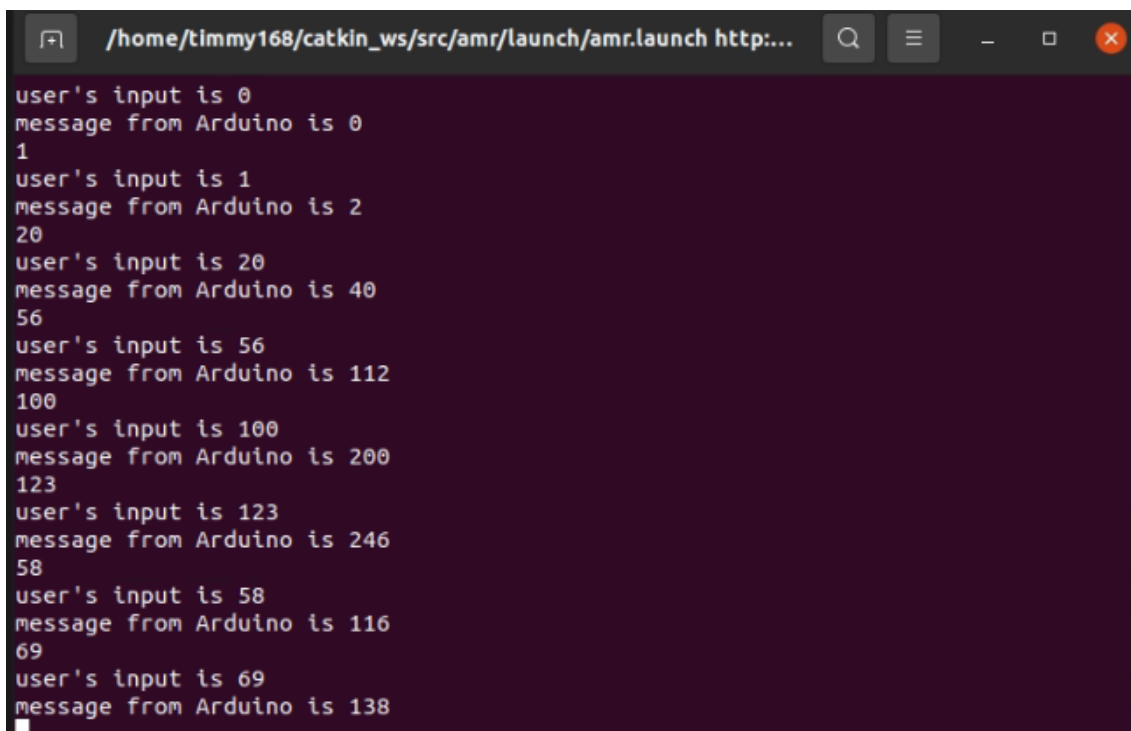
void setup()
{
    nh.initNode();
    nh.advertise(publisher);
    nh.subscribe(subscriber);
}

void loop()
{
    nh.spinOnce();
    delay(1);
}

```

3. Result:

From the figure below is the result at the Ubuntu terminal, from the figure below we could clearly see each of the user's input and the message return from the Arduino. From the result we could see that the message has successfully multiplied by 2 and return from the Arduino.



```

/home/timmy168/catkin_ws/src/amr/launch/amr.launch http:...
user's input is 0
message from Arduino is 0
1
user's input is 1
message from Arduino is 2
20
user's input is 20
message from Arduino is 40
56
user's input is 56
message from Arduino is 112
100
user's input is 100
message from Arduino is 200
123
user's input is 123
message from Arduino is 246
58
user's input is 58
message from Arduino is 116
69
user's input is 69
message from Arduino is 138

```

Figure (1): The result on the Ubuntu terminal.

4. Discussion:

In this checkpoint, we've established a robust foundation for our project by ensuring the proper configuration of our development environment. The steps

taken are pivotal in setting up a smooth workflow for future development.

Operating System Installation: The installation of the Ubuntu Mate operating system provides a reliable and well-supported environment for our robotics project. This choice not only offers the familiarity of Ubuntu but also tailors the system for compatibility with ROS.

ROS Integration: The successful installation of ROS Melodic on the Raspberry Pi is a crucial milestone. ROS, as a flexible and versatile framework, equips us with the tools needed for robotics development. The "rosversion" command serves as a confirmation that the correct ROS version is in use, ensuring compatibility with our intended packages and libraries.

Communication Setup: Utilizing the ROS roserial package bridges the gap between our Raspberry Pi and Arduino. This bi-directional communication channel is essential for sending and receiving data between the two devices. In particular, our design leverages publishers and subscribers to facilitate seamless data exchange.

Topic-Based Communication: The use of topics simplifies data exchange between our components. By dedicating topics such as "num" for transmitting values to the Arduino and "multiple" for receiving the multiplied results, we establish a structured and organized communication framework.

Program Architecture: We've outlined the architectural design of our programs for both the Raspberry Pi and Arduino. These programs incorporate publishers and subscribers, ensuring that data flows efficiently between the devices. The Raspberry Pi's program handles message transmission to the Arduino and receives the processed results, while the Arduino's program executes the multiplication operation and relays the outcome.

Launch Script: Our implementation incorporates a launch file, streamlining the execution of multiple scripts within the terminal. This automation simplifies the setup process and enhances the user experience.

In conclusion, this checkpoint not only lays the groundwork for effective communication between the Raspberry Pi and Arduino but also establishes a structured and organized development environment. The successful configuration and integration of these components pave the way for the implementation of more advanced robotics functionalities in subsequent phases of our project.