Checkpoint # 2   Report

1. Purpose:

The purpose of this checkpoint is to ensure the precise setup of the motor on the mobile robot's base. This setup enables users to remotely control the robot via SSH, sending PWM analog signals to the motor. This functionality allows for various motor control actions, including moving forward, backward, turning left and right, and traversing a straight path with ease.

2. Description of Design:

This checkpoint encompasses two key tasks: (1) **Motor Setup**: The first task involves the precise installation of the LN298 motor on the mobile robot's base, ensuring that it is securely and correctly positioned. (2) **Remote Control and PWM Signaling:** The second task focuses on enabling remote control of the RosberryPi via SSH. This control mechanism involves sending PWM analog signals to the Arduino board, which, in turn, translates these signals into motor control actions.

In this meticulously designed process, we've established a system that utilizes publishers and subscribers, centered around a single topic. This topic serves as the conduit for users to input velocity messages to the Arduino. The message type is an array of INT32, with the 0th element managing the right wheel velocity and the 1st element controlling the left wheel velocity.

On the Raspberry Pi, we've crafted a program featuring a publisher responsible for dispatching velocity messages under the "velocity" topic to the Arduino. Simultaneously, a subscriber on the Raspberry Pi tunes into the "velocity_feedback" topic, receiving PWM values from the Arduino.

The Arduino's design mirrors this structure, encompassing both a publisher and a subscriber. The Arduino subscribes to the "velocity" topic from the Raspberry Pi, extracting the velocity values, executing signal transmission operations, and ultimately publishing the resulting values back to the Raspberry Pi via the "velocity_feedback" topic.

Following the construction of these programs within the ROS environment for the Raspberry Pi, we've orchestrated a launch file. This launch file streamlines the execution of multiple scripts within the terminal, simplifying the setup and enhancing the user experience.

With this groundwork laid, we will now delve into the intricate implementation details of this system.

Raspberry Pi:

(1)velocity_publisher.cpp

```
#include "ros/ros.h"
#include "std_msgs/Int32MultiArray.h"
#include <iostream>

class control : public ros::NodeHandle
{
public:
    control()
    {
        publisher = this-
>advertise<std_msgs::Int32MultiArray>("velocity", 10);
        velocity.data.resize(2); //0 for right, 1 for left
        velocity.data[0] = 0;
        velocity.data[1] = 0;
    }
    void setVelocity(int right, int left)
    {
        velocity.data[0] = right;
        velocity.data[1] = left;
    }
    void publishVelocity()
    {
        publisher.publish(velocity);
        std::cout<<"user's right is "<< velocity.data[0] <<std::endl;
        std::cout<<"user's left is "<< velocity.data[1] <<std::endl;
    }
private:
    std_msgs::Int32MultiArray velocity;
    ros::NodeHandle nh;
    ros::Publisher publisher;
};

int main(int argc, char **argv)
{
    ros::init(argc, argv, "velocity_publisher");
    auto node = std::make_shared<control>();
    while (ros::ok())
    {
        int right,left;
        // std::cout << "----------------------------------------" <<
std::endl;
        std::cout << "Enter user's right input or enter -1 to quit: "
<< std::endl;
        std::cin >> right;
        std::cout << "Enter user's left input or enter -1 to quit: " <<
std::endl;
        std::cin >> left;
        if(right == -1 || left == -1) ros::shutdown();
        else
        {
            node->setVelocity(right,left);
        }
        node->publishVelocity();
    }
    //ros::spin();
    return 0;
}
```

(2)velocity_subscriber.cpp

```
#include "ros/ros.h"
#include "std_msgs/Int32MultiArray.h"
#include <iostream>

class control : public ros::NodeHandle
{
public:
    control()
    {
        std::cout << "Start to subscribe!" << std::endl;
        subscriber = this-
>subscribe<std_msgs::Int32MultiArray>("velocity",
                                                                   10,
&control::VelocityCallback, this);
        velocity.data.resize(2);
    }

    void VelocityCallback(const std_msgs::Int32MultiArray::ConstPtr&
velocity_ptr)
    {
        velocity.data = velocity_ptr -> data;
        // std::cout<<"------------------------"<< std::endl;
        std::cout<<"right velocity from Arduino is "<< velocity.data[0]
<<std::endl;
        std::cout<<"left velocity from Arduino is "<< velocity.data[1]
<<std::endl;
    }
private:
    // ros::NodeHandle nh;
    ros::Subscriber subscriber;
    std_msgs::Int32MultiArray velocity;
};

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

(3)velocity.launch

```
<launch>
    <node name="velocity_publisher" pkg="amr"
type="velocity_publisher" output="screen" />
```

```
    <node name="velocity_sublisher" pkg="amr"
type="velocity_subscriber" output="screen" />
    <node name="rosserial_node" pkg="rosserial_python"
type="serial_node.py" output="screen">
    <param name="port" value="/dev/ttyACM0" /> </node>
</launch>
```

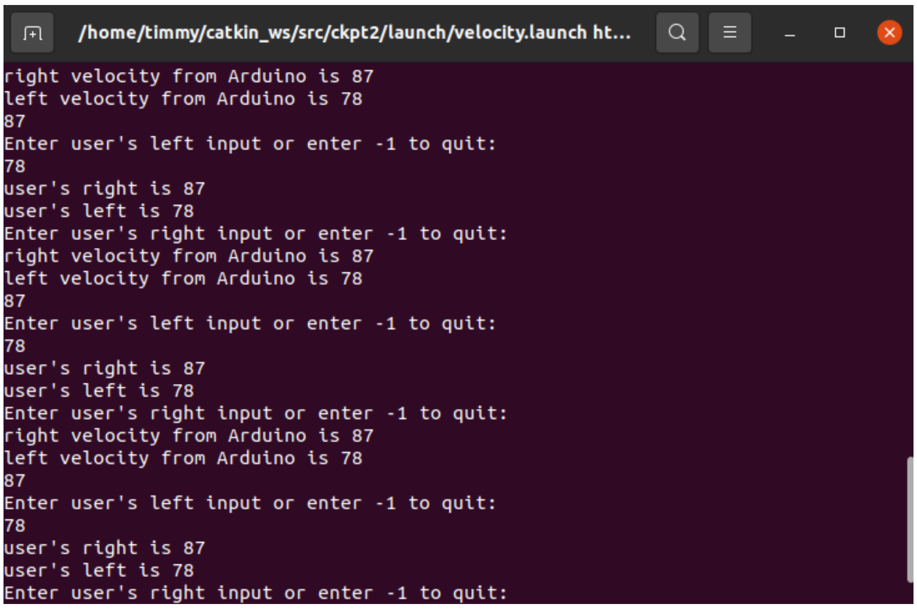
Arduino:

(1) checkpoint2.ino

```
#include<ros.h>
#include<std_msgs/Int32MultiArray.h>

#define EnA 5
#define EnB 6
#define In1 8
#define In2 9
#define In3 10
#define In4 11

ros::NodeHandle nh;
std_msgs::Int32MultiArray velocity;
ros::Publisher publisher("velocity_feedback", &velocity);
int pwm_r,pwm_l;

void move_robot(int right,int left)
{
  if(right>0)
  {
    digitalWrite(In1, LOW);
    digitalWrite(In2, HIGH);
  }
  else
  {
    digitalWrite(In1, HIGH);
    digitalWrite(In2, LOW);
  }
  if(left>0)
  {
    digitalWrite(In3, LOW);
    digitalWrite(In4, HIGH);
  }
  else
  {
    digitalWrite(In3, HIGH);
    digitalWrite(In4, LOW);
  }
  analogWrite(EnA, right);
  analogWrite(EnB, left);
}

void velocityCb(const std_msgs::Int32MultiArray& received_velocity)
{
    velocity.data = received_velocity.data;
    pwm_r = velocity.data[0];
    pwm_l = velocity.data[1];
    publisher.publish(&velocity);

}

ros::Subscriber<std_msgs::Int32MultiArray> subscriber("velocity",
&velocityCb);

void setup()
{
    nh.initNode();
    nh.advertise(publisher);
    nh.subscribe(subscriber);
    Serial.begin(57600);
    pinMode(EnA, OUTPUT);
    pinMode(EnB, OUTPUT);
    pinMode(In1, OUTPUT);
    pinMode(In2, OUTPUT);
    pinMode(In3, OUTPUT);
    pinMode(In4, OUTPUT);
}

void loop()
{
  move_robot(pwm_r,pwm_l);
  nh.spinOnce();
  delay(100);
}
```

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 velocity and the velocity feedback from the Arduino.


/home/timmy/catkin_ws/src/ckpt2/launch/velocity.launch ht...

```
right velocity from Arduino is 87
left velocity from Arduino is 78
87
Enter user's left input or enter -1 to quit:
78
user's right is 87
user's left is 78
Enter user's right input or enter -1 to quit:
right velocity from Arduino is 87
left velocity from Arduino is 78
87
Enter user's left input or enter -1 to quit:
78
user's right is 87
user's left is 78
Enter user's right input or enter -1 to quit:
right velocity from Arduino is 87
left velocity from Arduino is 78
87
Enter user's left input or enter -1 to quit:
78
user's right is 87
user's left is 78
Enter user's right input or enter -1 to quit:
```

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

4. Discussion:

This work serves the critical purpose of establishing precise motor configuration on the mobile robot's base. The successful setup empowers users to exercise remote control over the robot via SSH, allowing them to transmit PWM analog signals to the motor. This, in turn, enables a spectrum of motor control actions, encompassing forward and backward movements, as well as smooth turns to the left and right.

The work is comprised of two key objectives:

(1) **Motor Setup**: The first task is to ensure the accurate positioning and secure installation of the LN298 motor on the mobile robot's base. This critical step sets the foundation for reliable motor control.

(2) **Remote Control and PWM Signaling**: The second objective is to establish a mechanism for remote control of the RosberryPi through SSH. This entails sending PWM analog signals to the Arduino board, which then translates these signals into specific motor control actions.

The meticulously designed process revolves around a system built on the principles of publishers and subscribers, centering around a single topic. This topic serves as the conduit for users to input velocity messages to the Arduino. The message type utilized is an array of INT32, with the 0th element governing the right wheel's velocity, and the 1st element controlling the left wheel's velocity.

For the Raspberry Pi, a program has been crafted featuring a publisher responsible for transmitting velocity messages under the "velocity" topic to the Arduino. Simultaneously, a subscriber on the Raspberry Pi subscribes to the "velocity_feedback" topic, which receives PWM values from the Arduino.

In the Arduino's design, the structure mirrors that of the Raspberry Pi, encompassing both a publisher and a subscriber. The Arduino subscribes to the "velocity" topic from the Raspberry Pi, retrieving the velocity values, executing the signal transmission operations, and ultimately publishing the resulting values back to the Raspberry Pi under the "velocity_feedback" topic.

To streamline the execution of this system within the ROS environment for the Raspberry Pi, a launch file has been thoughtfully engineered. This launch file simplifies the setup by enabling the execution of multiple scripts within the terminal, thus enhancing the user experience.

With the foundational work completed, the next phase involves the sensors to use for more function to be implemented.