

ECE 510 : Intelligent Robotics III

Countess Quanta - using ROS

NavatejaReddy Kothakapu
PSU ID : 907013050

Table of Content

1. All About Countess Quanta	Page 3
2. ROS with Countess Quanta	3
3. Pioneer Base Navigation	4
3.1 Pioneer Server	5
3.2 Pioneer Client	8
4. Arm	9
4.1 Arm Connections Schematic	10
4.2 ROS serial - Arduino Code	10
5. All Together - Testing	15

I recommended to go through the ROS tutorials before going through this documentation because i didn't not explained package creation, building and some other core concepts which are explained clearly on ROS wiki and these skills are needed for clear understanding of this documentation

Thank you Professor Perkowski and Melhi for your support and help.

All the source code discussed in this documentation is available through GitHub. You can download at <https://github.com/ntej/CountessQuanta-ROS>

1 | All About Countess Quanta

Countess Quanta is a mobile robot. Its base consists of wheels which is a Pioneer robot, and can be controlled and navigated independently.

With the recent modification of its right arm, Countess Quanta is equipped to pick a small objects such as soda cans and medicine bottle from the floor. For now the object position on the ground is given manually by the user but in future the object position is detected using a depth camera.

2 | ROS with Countess Quanta

In this section we will learn how Countess Quanta control is implemented using ROS

The Countess Quanta has three main modules(nodes in ROS)....

Pioneer base - For the navigation of the robot

ARM - Right arm for picking up the objects

Intel Real Sense - Depth Camera for detecting the objects position(not yet successfully implemented but for now object position is giving manually using ROS client)

So now these three modules should intercommunicate each other, which is implemented using ROS. The ROS version we used is Indigo which is highly recommend for the stability. Before going forward on how this modules are implemented in ROS, lets have a look in to the nodes and topics graph of countess quanta generated by ROS rqt_graph package. See figure 2.0.0

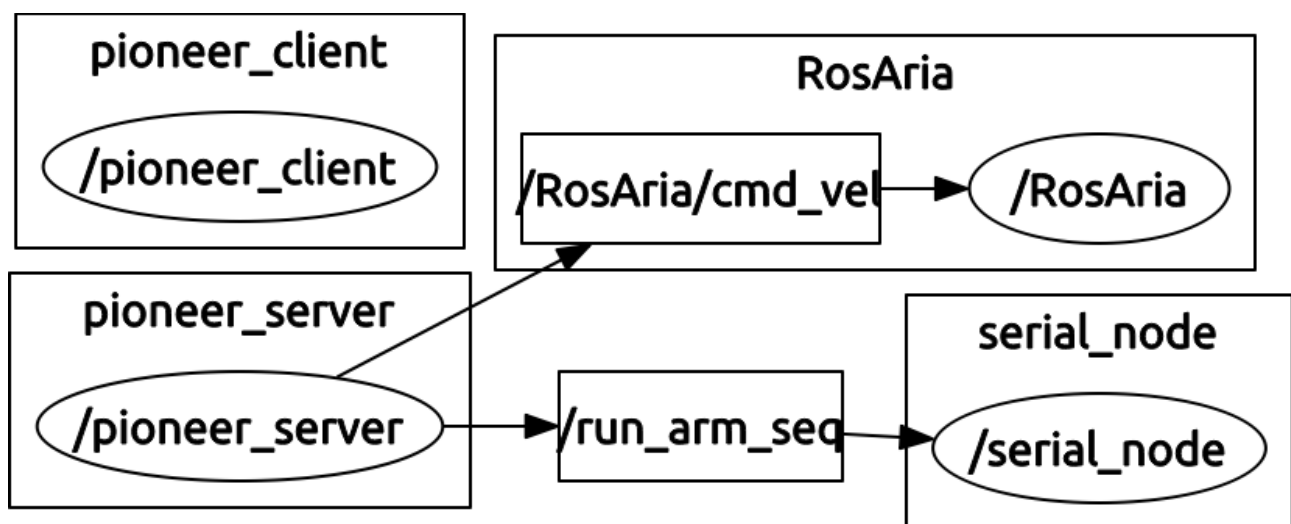


Fig 2.0.0 Nodes and Topics of countess Quanta

From the above figure you can see all the nodes and topics of countess Quanta in ROS. Lets learn about each node in detail.

Pioneer_server : This node is responsible for navigating the pioneer robot towards the object. It need object X and Z position to navigate the pioneer base. Have you wondered what are X and Z ?

More about X and Z are discussed in the next section. This node also initiate the arm pick up sequence by publishing Empty message on “run_arm_seq” topic of the Arduino node after moving the pioneer base to the object.

RosAria : RosAria is the node which controls the pioneer base using Aria libraries to which pioneer_server node is subscribed to topic ‘/RosAria/cmd_vel’. The pioneer sever publishes message of type geometry twist on ‘/RosAria/cmd_vel’ topic to which pioneer responds.

serial_node : It is the Arduino node which is responsible for the arm sequence of picking up the object. pioneer_server publishes empty message on ‘/run_arm_seq’ topic(after moving pioneer base to the object) which makes serial_node to run the arm sequence. Serial_node uses ‘rosterial’ protocol for wrapping standard ROS serialized messages and multiplexing multiple topics and services over a character device such as Arduino.

pioneer_client : We discussed earlier that the object position is determined by X and Z value which comes from the depth camera which is under development stage, so instead of death camera we use pioneer_client node which inputs X and Z values to the pioneer_server which in turn moves the pioneer base.

note: ‘pioneer_client’ and ‘pioneer_server’ are part of ‘pioneer’ ROS package.

3 | Pioneer Base Navigation

In this section we will learn how ROS service and client method is used to control the pioneer base. Before going through the pioneer package source code, lets learn how pioneer is navigated towards the object, see the figure 3.0.0

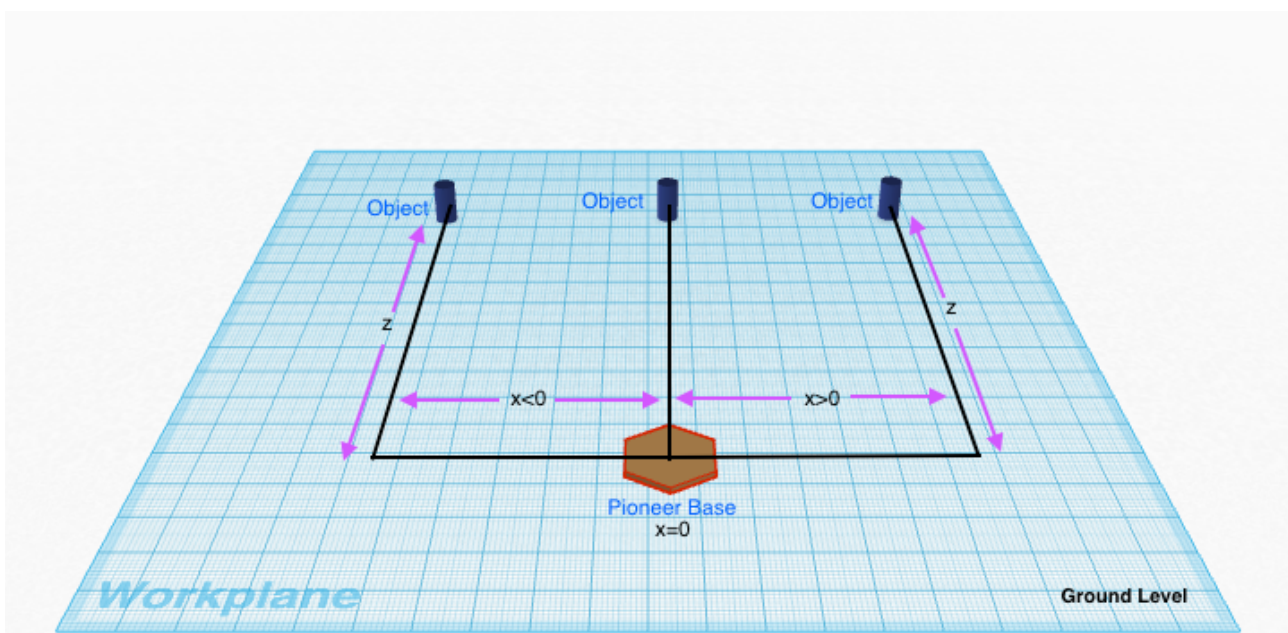


Fig 3.0.0 Pioneer navigation

Source: built on <https://thinkercad.com>

From the above figure X is the horizontal distance(in meters) of pioneer base from the detected object i.e when $x > 0$ the object is on right side with respect to pioneer base and vice versa, when $x = 0$ the object is straight in front of the pioneer base. Z(in meters) is the vertical distance of the object from the pioneer base which is always positive and greater than zero. Now you know what are X and Z values of the detected object, again which should come from the depth camera like intel real sense but as it is in developing stage so X and Z values are given manually through pioneer_client.

3.1 Pioneer Server

Pioneer server is the ROS service which takes request and gives response. Pioneer server takes X and Z values as request and moves the pioneer to the requested position and responds with success message, next it publishes empty message on '/run_arm_seq' topic which makes Arduino node to execute the arm pick up sequence. Now let's go through the pioneer_server code, the code is well commented so it's easy to understand.

```
/*Author : NavatejaReddy*/
#include "ros/ros.h"
#include "pioneer/MovePioneer.h"
#include <geometry_msgs/Twist.h>
#include <std_msgs/Empty.h>
#include <unistd.h> //header for time delay function usleep()

#define LINEAR_VELOCITY 0.1 //sets linear velocity
#define RIGHT_ANGULAR_VELOCITY -0.1 //sets angular velocity
#define LEFT_ANGULAR_VELOCITY 0.1 //sets angular velocity
#define ANGLE_IN_DEGREES 1.5708 //defines the angle(in degrees) to turn the pioneer
#define SYSTEMATIC_ERROR 1.03 /*to correct error of distance covered with defined linear
velocity by multiplying with temptime measured during runtime */

ros::Publisher pub;
ros::Publisher arduino_pub;

bool move(pioneer::MovePioneer::Request &req, pioneer::MovePioneer::Response &res)
{
    float x_position = req.xpos;
    float z_position = req.zpos;
    double temptime;

    geometry_msgs::Twist msg;
    std_msgs::Empty arduino_msg;
    std::cerr << "Attempting to move: ";
    /*-----object is on right side of robot-----*/
```

```

if(x_position>0)
{
    //step 1 : turn 90 towards right
    std::cerr << "X > 0" << std::endl;
    msg.angular.z = RIGHT_ANGULAR_VELOCITY ; //setting the pioneer angular velocity
    pub.publish(msg); //publishing the msg
    usleep(abs(ANGLE_IN_DEGREES/RIGHT_ANGULAR_VELOCITY)*1000000);
    //calculated time to turn givendegrees with given angular velocity
    msg.angular.z = 0 ; //setting the pioneer angular velocity to 0 i.e stopping
    pub.publish(msg);
    usleep(500000); //0.5 sec delay

    //step 2 : move along x_position
    msg.linear.x = LINEAR_VELOCITY ; //setting the pioneer linear velocity
    pub.publish(msg);
    temptime = x_position/LINEAR_VELOCITY ; /*calculating the time to travel the x_position by
pioneer base*/
    usleep(abs(temptime*SYSTAMATIC_ERROR*1000000)); //dealy to allow pioneer to travel x_position
    msg.linear.x = 0; //setting the pioneer linear velocity to 0 i.e stopping
    pub.publish(msg);
    usleep(500000);

    //step 3 : turn 90 towards left
    msg.angular.z = LEFT_ANGULAR_VELOCITY ;
    pub.publish(msg);
    usleep(abs(ANGLE_IN_DEGREES/LEFT_ANGULAR_VELOCITY)*1000000);
    msg.angular.z = 0 ;
    pub.publish(msg);
    usleep(500000);

    //step 4 : move along z_position
    msg.linear.x = LINEAR_VELOCITY ;
    pub.publish(msg);
    temptime = z_position/LINEAR_VELOCITY;

    usleep(abs(temptime*SYSTAMATIC_ERROR*1000000));
    msg.linear.x = 0;
    pub.publish(msg);
}

/*-----object is on left side of robot-----*/

```

```

else if(x_position < 0)
{
    std::cerr << "X < 0 " << std::endl;

    //step 1 : turn 90 towards left
    msg.angular.z = LEFT_ANGULAR_VELOCITY ;
    pub.publish(msg);
    usleep(abs(ANGLE_IN_DEGREES/LEFT_ANGULAR_VELOCITY)*1000000);
    msg.angular.z = 0 ;
    pub.publish(msg);
    usleep(500000);

    //step 2 : move along x_position
    msg.linear.x = LINEAR_VELOCITY ;
    pub.publish(msg);
    temptime = x_position/LINEAR_VELOCITY ;
    usleep(abs(temptime*SYSTEMATIC_ERROR*1000000));
    msg.linear.x = 0;
    pub.publish(msg);
    usleep(500000);

    //step 3 : turn 90 towards right
    msg.angular.z = RIGHT_ANGULAR_VELOCITY ;
    pub.publish(msg);
    usleep(abs(ANGLE_IN_DEGREES/RIGHT_ANGULAR_VELOCITY)*1000000);
    msg.angular.z = 0 ;
    pub.publish(msg);
    usleep(500000);

    //step 4 : move along z_position
    msg.linear.x = LINEAR_VELOCITY ;
    pub.publish(msg);
    temptime = z_position/LINEAR_VELOCITY;
    usleep(abs(temptime*SYSTEMATIC_ERROR*1000000));
    msg.linear.x = 0;
    pub.publish(msg);
}

/*-----object is facing straight to the robot-----*/
else
{

```

```

std::cerr << "X == 0" << std::endl;
msg.linear.x = LINEAR_VELOCITY ;
pub.publish(msg);
temptime = z_position/LINEAR_VELOCITY;
usleep(abs(temptime*SYSTAMATIC_ERROR*1000000));
msg.linear.x = 0;
pub.publish(msg);
}

res.result = "Success! Moved the pioneer base to the object,now running Arm Sequence";
arduino_pub.publish(arduino_msg);

return true;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "pioneer_server");
    ros::NodeHandle n;
    pub = n.advertise<geometry_msgs::Twist>("/RosAria/cmd_vel",100); /* "packagename*/cmd_vel"
    topic with message type "geometry_msgs/twist"|subscribing to topic*/
    arduino_pub=n.advertise<std_msgs::Empty>("run_arm_seq",100); //subscribing to arduino node
    ros::ServiceServer service = n.advertiseService("move_pioneer",move);
    ROS_INFO("Ready to move pioneer base, Give xpos and zpos of the detected object");
    ros::spin();
    return 0;
}

```

3.2 Pioneer Client

As discussed earlier pioneer client provides X and Z values as request to the pioneer server which moves to the object. Now lets go through the pioneer_client code.

```

/*Author:NavatejaReddy*/
#include "ros/ros.h"
#include "pioneer/MovePioneer.h"
#include <geometry_msgs/Twist.h>
#include <cstdlib>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "pioneer_client");

```



```

if(argc !=3)
{
    ROS_INFO("usage: pioneer_client xpos zpos");
    return 1;
}

ros::NodeHandle n;
ros::ServiceClient client = n.serviceClient<pioneer::MovePioneer>("move_pioneer");
pioneer::MovePioneer srv;

srv.request.xpos = atoll(argv[1]);
srv.request.zpos = atoll(argv[2]);

if(client.call(srv))
{
    ROS_INFO("%s",srv.response.result.c_str());
}
else
{
    ROS_ERROR("failed to call the service pioneer_server ");
    return 1;
}

return 0;
}

```

4 | Arm

The arm is build using 2 DC motor and 4 servos. The DC motors are used to control the shoulder and upper arm, lower arm is built using servos. See the figure 4.0.0

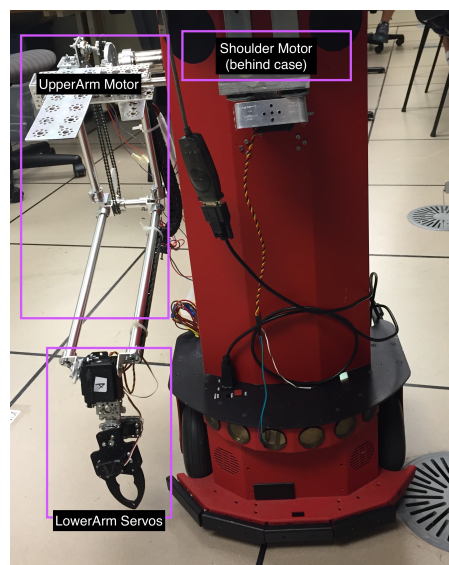


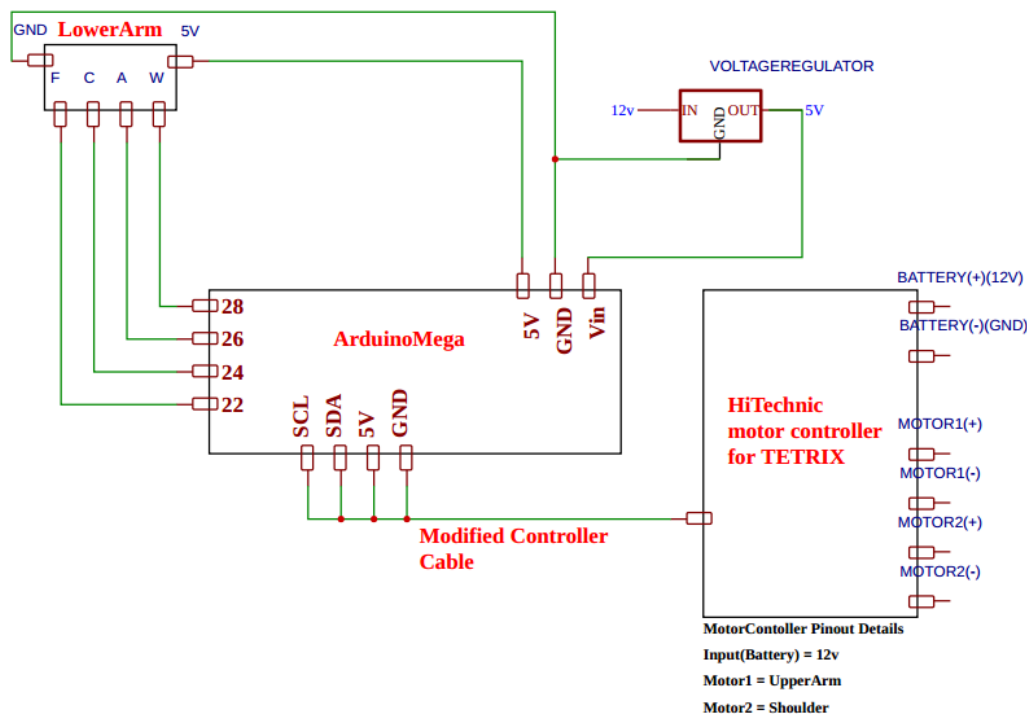
Fig 4.0.0 Countess Quanta Arm

The arm is controlled using Arduino. The DC motors are interfaced to Arduino with the help Hi-technic DC motor controller which communicates with Arduino using I2C protocol. To learn more about arm, refer saroj_final_doc.pdf on Github. See figure 4.1.0 for the Arm connections schematic.

4.1 Arm Connections Schematic

CountessQuanta Arm Connections Schematic

Note : The 12V input to the Voltage Regulator and Motor controller is supplied from PioneerBase batteries



NavatejaReddy(navatejareddykothakapu@gmail.com)

Fig 4.1.0

Source: built on <https://easyeda.com>

4.2 ROS Serial - Arduino Arm Control

Countess Quanta arm is controlled using Arduino and Arduino communicates with ROS network using rosserial protocol. Coming to the control part, Arduino uses potentiometer as feedback to control the position of the lower arm motor and shoulder motor works just in power mode with two increments(loop) forward before picking up the object and two increments(loop) backwards after picking up the object. Lets go through the Arduino sketch before proceeding to next section.

```

/*Author : Navatejareddy */
#include <Wire.h>
#include <Servo.h>
#include <ros.h>
#include <std_msgs/Empty.h>

#define FINGER_SERVO_POS      22      //'F'
#define WRISTS2S_SERVO_POS    24      //'C'
#define WRISTUPDOWN_SERVO_POS 26      //'A'
#define WRISTROTATION_SERVO_POS 28    //'W'
#define INIT_FINGERS 123
#define INIT_WRIST_UD 75
#define INIT_WRIST_SIDE2SIDE 60
#define INIT_WRIST_ROTATE 70

Servo fingers;
Servo wristRotation;
Servo wristSide2Side;
Servo wristUpDown;

#define DEVICE_ADDRESS 0x05
#define M1_MODE_REGISTER 0x44
#define M2_MODE_REGISTER 0x47
#define M1_POWER_REGISTER 0x45
#define M2_POWER_REGISTER 0x46

int initialPotVal;
int potVal;
bool iteration = true;

ros::NodeHandle nh;
void messageCb(const std_msgs::Empty& arm_seq_msg)
{
    if(iteration == true) //arm sequence
    {
        moveToFinalPos();
        delay(2000);
        moveToInitialPos();
    }
}

ros::Subscriber<std_msgs::Empty> sub("run_arm_seq",&messageCb);

```

```

void setup() {
Wire.begin();
Serial.begin(9600);

/*****servo*****/
// initialize servos and attaces to pins
fingers.attach(FINGER_SERVO_POS);
wristRotation.attach(WRISTROTATION_SERVO_POS);
wristSide2Side.attach(WRISTS2S_SERVO_POS);
wristUpDown.attach(WRISTUPDOWN_SERVO_POS);
delay(200); // waits to be assign

//Initialization
wristUpDown.write(INIT_WRIST_UD);
delay(1000);
fingers.write(INIT_FINGERS);      // Finger Motor range: 90 (close) to 180 (open)
delay(1000);
wristRotation.write(INIT_WRIST_ROTATE); /* Wrist Motor range: 15 (out) to 180 (in)   and 145 is
neutral*/
delay(1000);
wristSide2Side.write(INIT_WRIST_SIDE2SIDE); // Carpel Motor range : 0 (in) t0 90 (out)
delay(1000);

/*****Motor*****/
initialPotVal = analogRead(A6);
Wire.beginTransmission(DEVICE_ADDRESS); //connecting to slave i.e DC motor controller
Wire.write(M1_MODE_REGISTER); //Writing to mode register, i.e power mode
Wire.write(0x00);
Wire.endTransmission(0);
delay(1000);

Wire.beginTransmission(DEVICE_ADDRESS); //connecting to slave i.e DC motor controller
Wire.write(M2_MODE_REGISTER); //Writing to mode register, i.e power mode
Wire.write(0x00);
Wire.endTransmission(0);

nh.initNode();
nh.subscribe(sub);
}

```

```

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

void moveToFinalPos() //moving upperarm to lower position
{
    potVal = analogRead(A6);
    while(abs(potVal-initialPotVal)<100)
    {
        delay(1000);
        Wire.beginTransmission(DEVICE_ADDRESS);
        Wire.write(M1_POWER_REGISTER);
        Wire.write(-30);
        Wire.endTransmission(0);
        delay(100);

        Wire.beginTransmission(DEVICE_ADDRESS);
        Wire.write(M1_POWER_REGISTER);
        Wire.write(0);
        Wire.endTransmission(0);
        potVal = analogRead(A6);
    }

    fingers.write(75);
    delay(2000);

    for(int m2count=0;m2count<3;m2count++) //moving shoulder forward
    {
        delay(1000);
        Wire.beginTransmission(DEVICE_ADDRESS);
        Wire.write(M2_POWER_REGISTER);
        Wire.write(40);
        Wire.endTransmission(0);
        delay(100);

        Wire.beginTransmission(DEVICE_ADDRESS);
        Wire.write(M2_POWER_REGISTER);
        Wire.write(0);
        Wire.endTransmission(0);
    }
}

```

```

//servo code
wristUpDown.write(110); //making wrist straight
delay(2000);
fingers.write(123); //closing the fingers
delay(2000);
for(int m2count=0;m2count<2;m2count++) //bringing shoulder back
{
    delay(1000);
    Wire.beginTransmission(DEVICE_ADDRESS);
    Wire.write(M2_POWER_REGISTER);
    Wire.write(-40);
    Wire.endTransmission(0);
    delay(100);
    Wire.beginTransmission(DEVICE_ADDRESS);
    Wire.write(M2_POWER_REGISTER);
    Wire.write(0);
    Wire.endTransmission(0);
}
}

void moveToInitialPos() //moving upper arm to upper position
{
    potVal = analogRead(A6);
    while(potVal<initialPotVal-20)
    {
        delay(1000);
        Wire.beginTransmission(DEVICE_ADDRESS);
        Wire.write(M1_POWER_REGISTER);
        Wire.write(100);
        Wire.endTransmission(0);

        delay(100);

        Wire.beginTransmission(DEVICE_ADDRESS);
        Wire.write(M1_POWER_REGISTER);
        Wire.write(0);
        Wire.endTransmission(0);

        potVal = analogRead(A6);
    }
    delay(2000);
}

```

```
wristUpDown.write(75);  
iteration = false;  
}
```

5 | All Together - Testing

Now its time for testing, lets do it step by step. First we will set up arduino and roserial, next we will setup pioneer base and finally run the pioneer server and client. File/folder names used below can be downloaded from GitHub(<https://github.com/ntej/CountessQuanta-ROS>)

Note : The arm should be in initial position before running the sequence sketch, which is the starting position of the arm. Initial position means the lower arm should be approximately perpendicular to the upper arm. You can use *motor_calibrator.ino* sketch to bring the arm to initial position

Step 1 : Initiate the ROS by executing following command

```
$roscore
```

Step 2 : Connect the Arduino and execute the following command to determine to which port it is connected

```
$dmesg | grep tty
```

Change the permission of the port by running following command

```
$sudo chmod 666 <port Arduino is connected>
```

Next upload the *i2cmotor_servo_contoller_ros.ino* Arduino sketch and next execute the following command in the terminal

```
$roslaunch roserial_python serial_node.py <port to which Arduino is connected>
```

After running the above command make sure roserial is working correctly by checking through terminal output

Step 3 : Connect the pioneer base and determine to which port it is connected and change permission as done in step 2. Then run following command in terminal to connect to pioneer base

```
$roslaunch rosaria RosAria _port:= <port to which pioneer base is connected>
```

Instead of connecting to actual pioneer base, we can connect to mobilesims simulator by executing following command

```
$roslaunch rosaria RosAria
```

Step 4 : Now Arduino is on ROS serial and pioneer base is connected, next step is we need to build the pioneer package, to do so copy the *pioneer* folder to your catkin workspace and run the following command to build the pioneer package

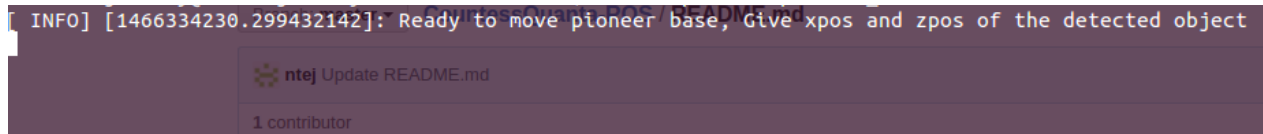
```
$catkin_make
```

Step 5 : After package building is completed, then execute the following command to start the pioneer server

```
$roslaunch pioneer pioneer_server
```

after running above command you will see output similar to shown below

```
[ INFO] [1466334230.299432142]: Ready to move pioneer base, Give xpos and zpos of the detected object
```



You can notice that pioneer server is waiting for x-position(xpos) and z-position(zpos) of the object to move the base which we give manually using pioneer client. As mentioned earlier xpos and zpos are the values which should come from depth camera(under development).

Step 6: Finally lets run the pioneer client by running the following command

```
$roslaunch pioneer_pioneer_client <xpos> <zpos>
```

after executing the above command the pioneer base moves to the object and then the pioneer server publishes empty message to the Arduino over rosserial, then Arduino executes the arm sequence of picking up the arm. Replace xpos and zpos with your desired x(meters) and z(meters) values in unsigned integer format(eg: **\$roslaunch pioneer_pioneer_client 1 3**)

Enjoy!

Reference and online tools used to make this documentation:

[1]<http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28c%2B%2B%29>

[2]<http://wiki.ros.org/rosterial>

[3]http://wiki.ros.org/rosterial_arduino/Tutorials/Blink

[4]"HiTechnic FIRST Motor Controller Specification", HiTechnic, 2016. [Online]. Available: <http://www.hitechnic.com/blog/wp-content/uploads/HiTechnic-Motor-Controller-Specification.pdf>. [Accessed: 21- Mar- 2016].

[5]Saroj Final Documentation from ECE: 579/479 Intelligent robotics II (Winter 2016)

[6]www.thinkercad.com

[7]www.easyeda.com