SCARA Arm

Yamaha YK800XGP

Wenhao Xu, Felix Zhao, Bolong Tan

This slide deck will be for our final presentation for ELEC391 SCARA ARM.
For team Yamaha YK800XGP.

Team Organisation

Wenhao Xu	Felix Zhao	Bolong Tan
Software Report Encoder PID control and simulink Simulation X	Hardware Report C-Code Circuit and PCB Design Microcontroller	System Report Solidworks Motors Actuators and Gearbox

For team organization, we split the work equally,
Wenhao took care of the software section
Felix took care of the hardware section
And Bolong took care of the system section.
We also collaborate and help each other on their parts whenever possible.

Electrical System RCG

Specification	Requirement	Constraint	Goal
Nominal Voltage	24V±2V	24V±1V	24V
Max Current	30A (both motor)	Max 50A	40A
Switching frequency	3kHz		Maximum
Size		140mm*100mm	
Circuit protection	Overcurrent Optical isolation		

For Electrical systems ECG, we need to design a circuit that will be able to support our motor selected.
We need to have a consistent nominal voltage and current that can satisfy our motors rated voltages.
We also need to not draw more current than we want from the wall plug to trigger the normally present wall plug fuses.
Our motor control board need to have a high switching frequency to reduce general component sizes and reduce our PCB size to be able to fit into our design's base cavity.
Lastly, of course we need basic circuit protection like overcurrent protections, voltage/current isolation, voltage/current surge protections and overheat protections.

Mechanical system RCG

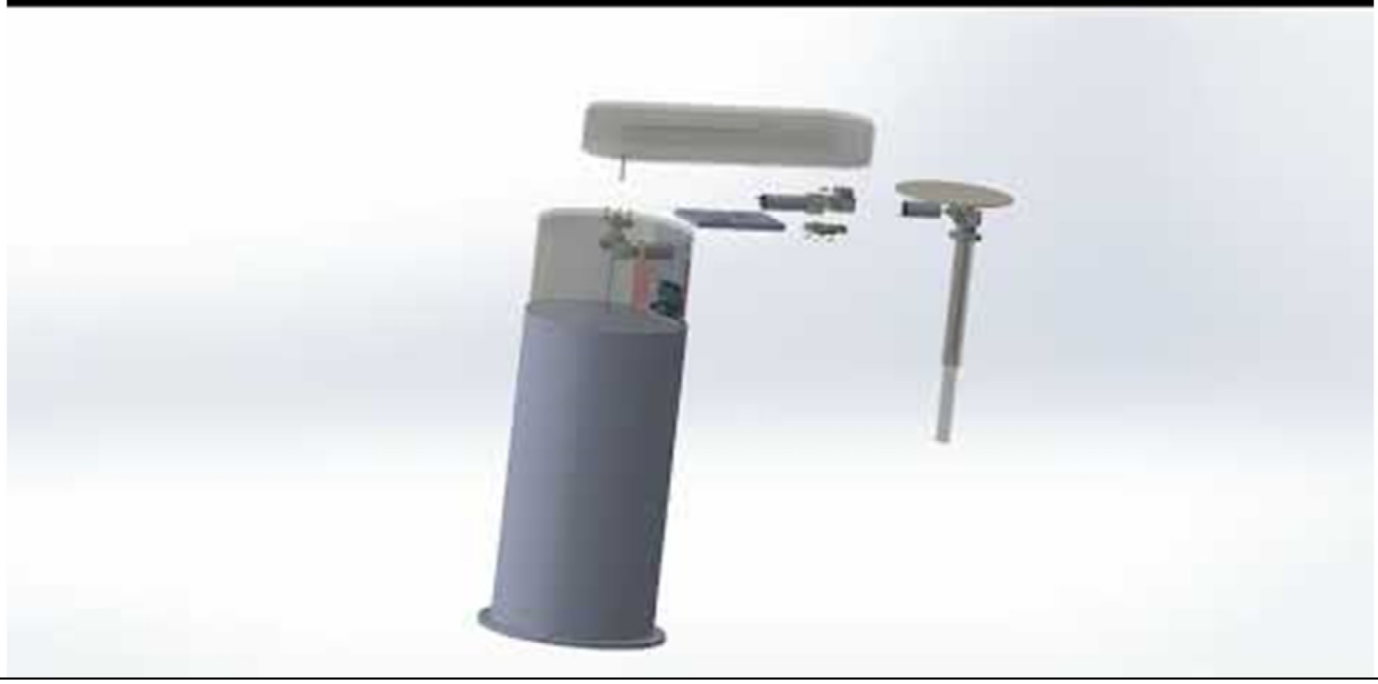
Specification	Requirement	Constraint	Goal
Work space diameter	1.2m	-	Maximum
Maximum height	550mm	-	Maximum
Payload	15Kg	-	Maximum
X,y speed	4.2 m/s		Minimum

This is our mechanical system RCG.
The work space diameter we want to achieve is 1.2m and the height is 550mm.
One of our robot arm is 350mm and the other one is 250mm in length.
The full length of our robot arm is 600mm. The work space diameter is 2*600mm= 1.2m.
The robot arm can reach any points within the work space.
For the payload, our goal is 15kg.
In our simulink and simulationx model,we used the 15kg load to simulate the whole model.
Even though the inertia due to the robot arm and 15Kg is pretty large.
Our design have successfully meet all these requirements.

Control System RCG

Specification	Requirement	Constraint	Goal
Steady State Error	< 2%	N/A	minimize
Overshoot	N/A	< 20%	minimize
Settle Time	< 1.5 sec	N/A	minimize
Rise Time	< 1 sec	N/A	minimize

This is our control system RCG. First we want an accurate arm, so we want to minimize steady state error and overshoot. We also want a fast arm, so we want to have good settle time and rise time.



As you can see this is our explode animation with the extra feature.

Our design consists of one base, three robot arms, three motors, three worm gears etc.

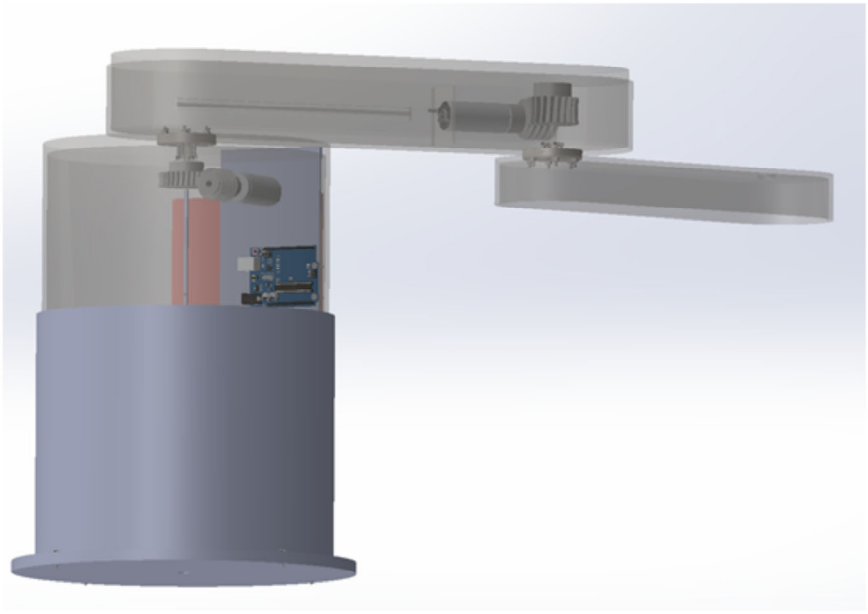
Two robot arms can move in the horizontal plane. Both arms can rotate 360 degree.

Our robot arm can also move in Z direction.

We used one extra motor and one extra worm gear.

1. Motor rotate shaft which is connected to worm gear (gear ratio is 30)
1. Worm gear is connected to the lead screw through the shaft
1. Worm gear and leads screws rotates at the same speed
1. A hex nut which connected to a rod is used
1. Hex nut cannot rotate because it is surrounded by a shell
1. Lead screw forces the hex nut to move up or down

Solidwork Model (Motor- Dcx 26L)



Worm and Gear Sold Separately

7

This is the solid work model we use without extra features whose arm can only move in horizontal plane.

Two robot arms can move in the horizontal plane. Both arms can rotate 360 degree.

The design consists of one base,two robot arms and two motors, two worm gears and etc.

The motor we chose is Dcx-26L.The reason we chose this motor is that:

1. Our robots arm length is 600mm which is quite long

1. We also want to pick a 15kg object

According to the calculation the torque we need is around 50Nm. The Dcx-26L motor is fits our requirement best.

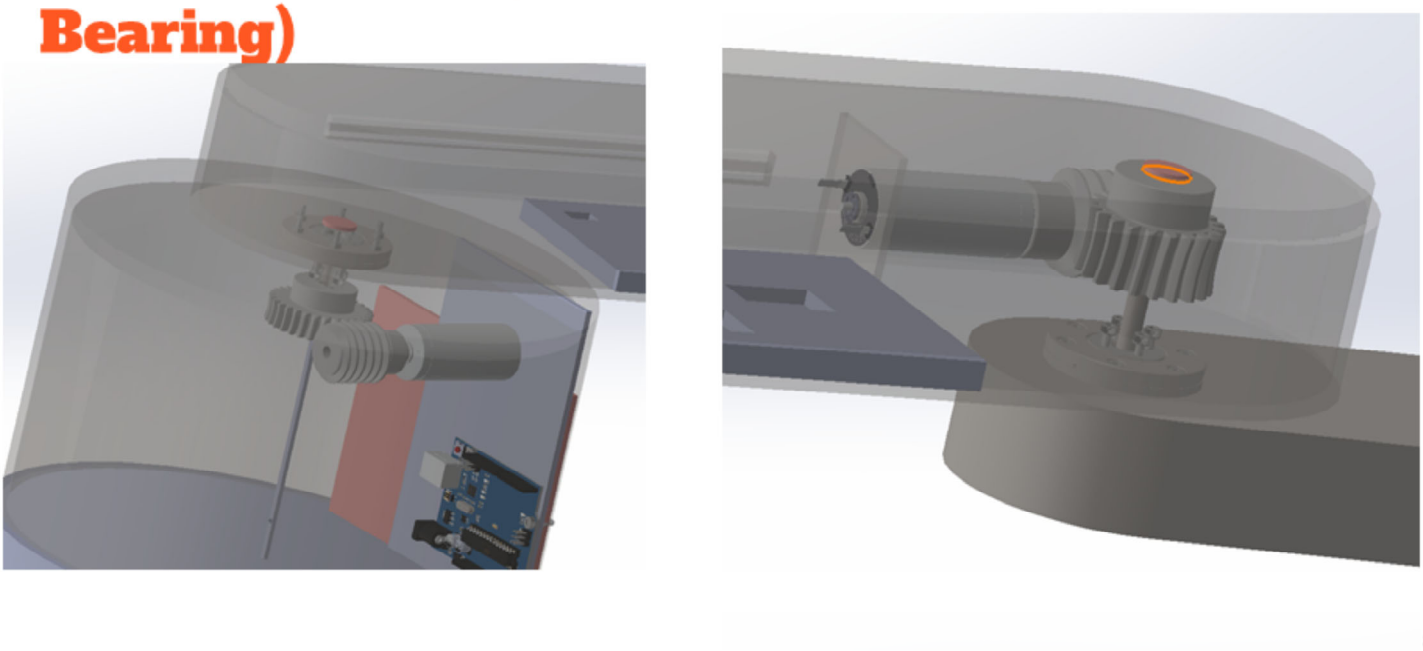
Gear ratio of two worm gears we used are 20 and 30.

The advantage of worm gear is that:

1. Cheaper

1. Do not require too much space (can be easily put into robot arm)

Shoulder Joint (Face-Mount Crossed-Roller Bearing)



8

Two face-mount crossed-Roller bearing are used to make sure both robot arms can rotate freely in horizontal plane.

One face-Mount crossed-Roller bearing is used to connect base to arm.

The other one is used to connect two arms together.

For one of face-mount crossed-Roller bearing:

The inner part of face-mount crossed-Roller bearing is connected to the base using screws and nuts.

The inner part is fixed with respect to base.

The outer part of face-mount crossed-Roller bearing is connected to the robot arm using screws and nuts. It rotates at the same speed with arm.

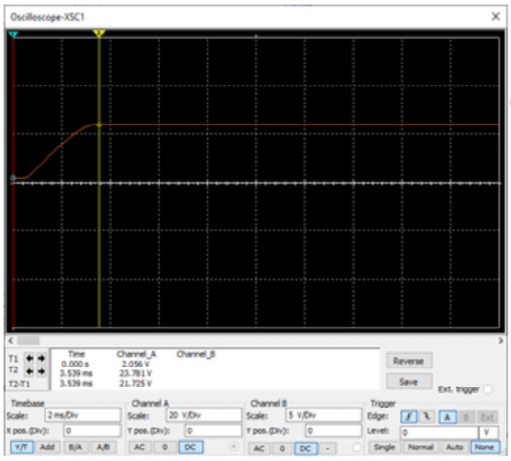
For the other face-mount crossed-Roller bearing:

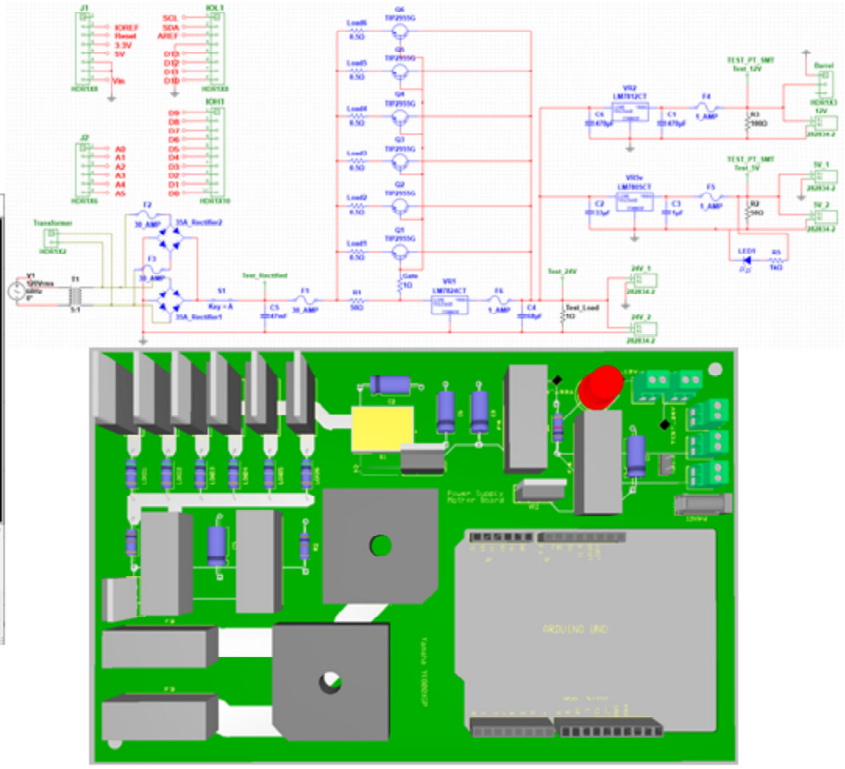
The inner part of face-mount crossed-Roller bearing is connected to the upper arm using screws and nuts.

The inner part is fixed with respect to upper arm..

The outer part of face-mount crossed-Roller bearing is connected to the lower robot arm using screws and nuts. It rotates at the same speed with lower arm.

Power Supply / Motherboard





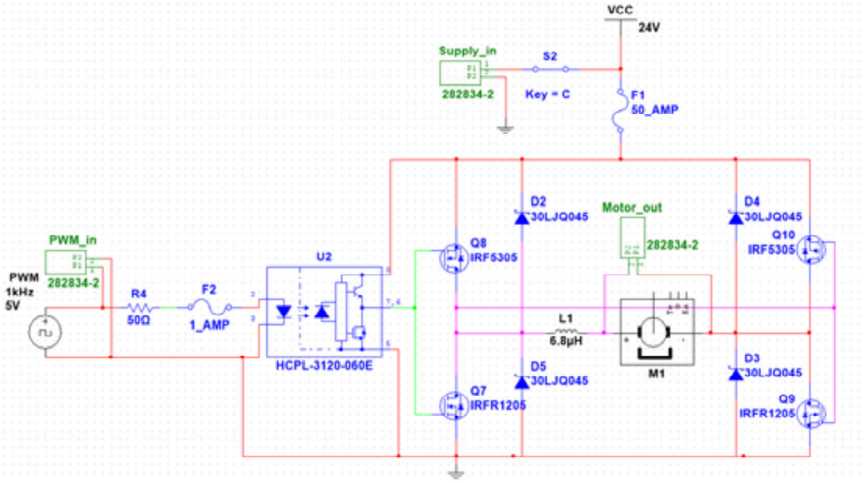
9

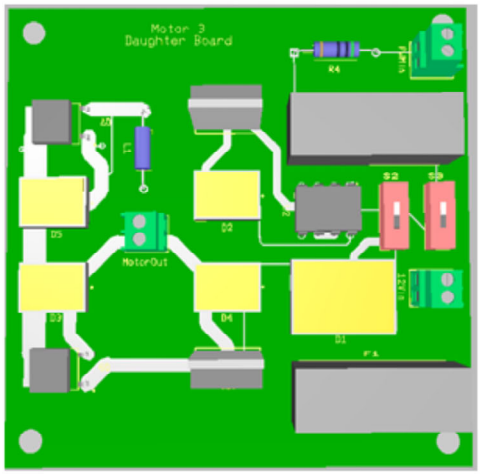
For our power supply board design, with the previously mentioned RCG in mind, we developed out board to have:

- 5V, 12V and 24V output support for all the external ICs we need. (Arduino, Motor, Sensor)
- 40A maximum current supplied to the motor output.
- Stable voltage output under high current applications. (24±0.3V under 25A, 24±1V under 40A)
- Safety fuses in case of current surges for each output and transformers to prevent meltdowns.
- LED indicators and test points for easy debugging.
- Different thickness traces for high/low current areas, mounting holes for installation of the board, and a slot for arduino board.

The resulted board is able to support 2 26L motors and all other needed ICs with ease and satisfies our predefined design RCGs.

Motor Driver/ Daughter Board





For our Motor Driver board or daughter boards, we designed our board to have:

- Only one PWM input needed to regulate the 24V input voltage utilizing the H-bridge circuit.
- Safety fuses in case of current surges for each input.
- Back emf protection using series of schottky diodes.
- Inrush current protection with the help of both the fuse and an inductor.
- An opto-coupler for current isolation for the arduino input side and ability to drive a 24V input with Arduino's 5V output.
- Different thickness traces for high/low current areas, mounting holes for installation of the board

The resulted board is able to drive our motors with relative accuracy and satisfies our predefined design RCGs.

Sensor and Homing sequence

```
void homing(double phase1, double phase2){ //input from the optical sensor
    while (phase1 <= phase2*1.01 && phase1 >= phase2*0.99) //allowing 1% error
    {
        while (phase1 < phase2) //if the motor is in the backward position
        {
            forward();
        }
        stopMotor();
        sleep(1000);
        while (phase1 > phase2) // if the motor is in the forward position
        {
            reverse();
        }
        stopMotor();
        sleep(1000);
    }
    homingdone = true; //assert homing done flag
}
```

11

Next, for our optical sensor, we chose to use a on-market on-shaft optical encoder from US Digital : E4P miniature.
It is able to return 2 sets of square waves with varying phases depending on the shaft's current angle from the home position.
With that in mind we can write a simple homing sequence like the one on the right:

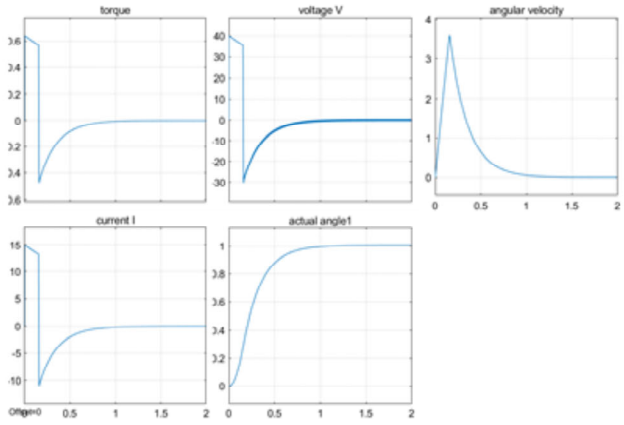
1. Check both phases input, see if the two input values are the same/are we currently at base position (1% error accepted)

2. If they are not the same, determine which direction we need to move to be back to the home position.

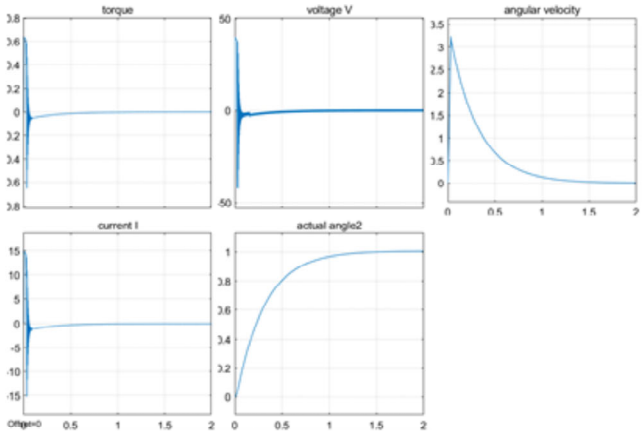
3. Initiate the motor movement sequence until we reached the home position.

4. Raise the done flag so the system only performs homing sequence once upon startup.

Simulink



Joint 1



Joint 2

12

Then we use simulink to model the control system and tune the PID controller. First we do 10 steps.
For the two motors:

- K = 0.4 Kp = 13.3 Ki = 1 Kd = 44.44 (joint 1)
- K = 0.4 kp = 8.7 Ki = 1 Kd = 18.9 (joint 2)

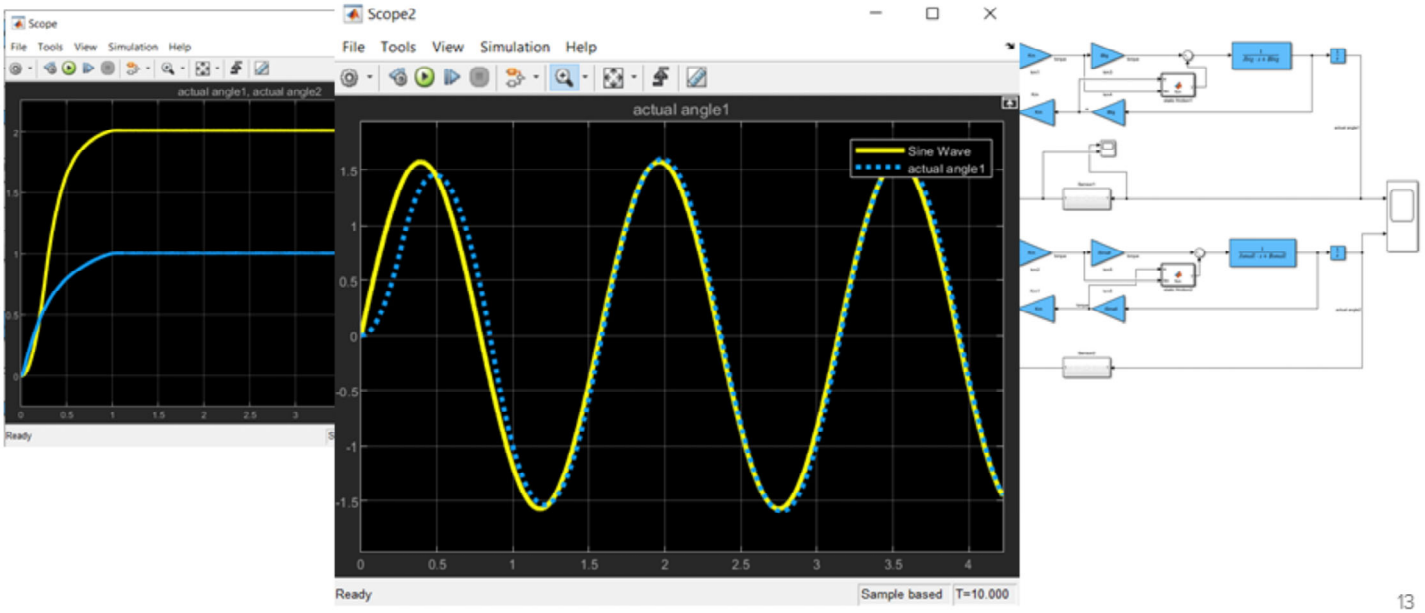
We would like to address the saturation introduced by the amplifier. After that, we do the heuristic tuning again and get

- K = 3 Kp = 50 Ki = 1 Kd = 10 (joint 1)
- K = 3 Kp = 40 Ki = 1 Kd = 10 (joint 2)

We have higher master to accommodate the saturation, and lowered the Kd to lower the overshoot to satisfy the RCG of minimizing overshoot, and raised Kp to make the motor have faster settle time (RCG).

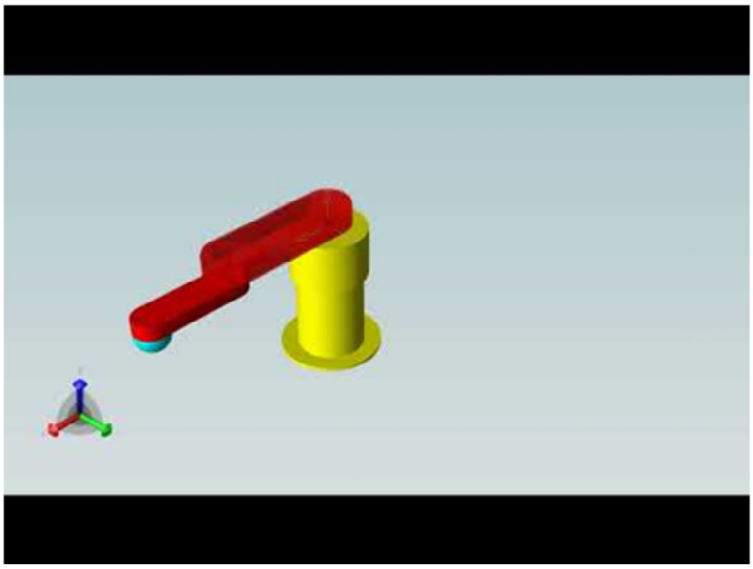
We can see the step responses settle in less than 1.5 seconds, which is good for our RCGs.

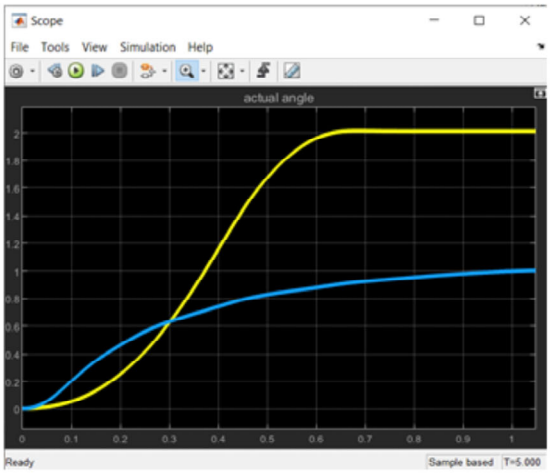
Simulink Cont'd



We add more nonlinearities such as the nonlinearity introduced by sensor and static friction. For sensor, the encoder we choose has 1440 max pulses per revolution, which means the relative sensor can detect 1/16 degrees of angle. We convert the actual angle to degrees, times 16 and use matlab function floor to emulate the sensor's behavior, then convert the whole number counts back to radians. For static friction we use a matlab function to make the actual angular velocity zero until the arms overcome the static friction. As the step response graph show, we still has a good settle time and low overshoot. Then we see if we can catch a moving target. As the result shows, we have a little overhead at start, but then we only have some lag after the target, which is good for our RCG of being fast.

Simulation X

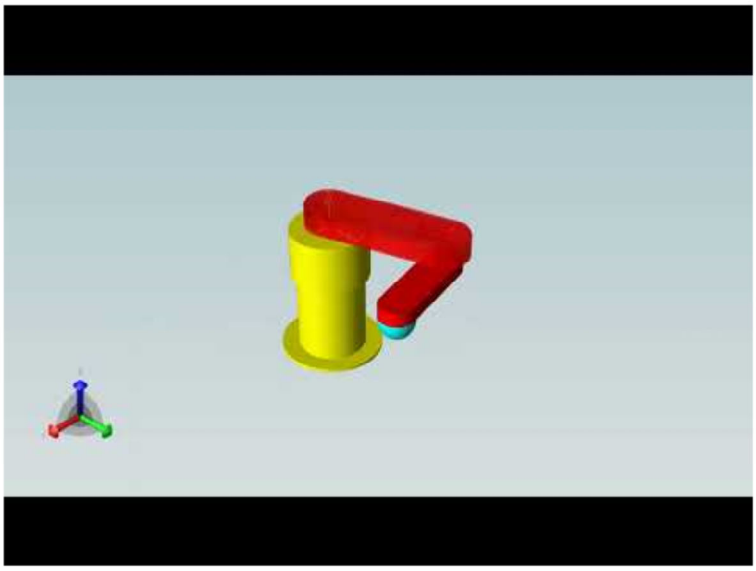


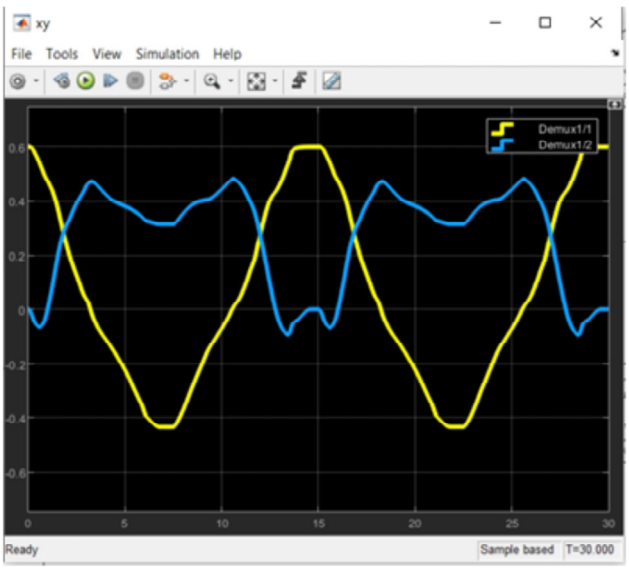


14

We then use Simulation X to simulate the mechanical system. We set up a co-simulation system which connected Matlab Simulink and Simulation X. This helps us better study the effect of some factors such as gearboxes and gravity. The video shows step response of both the forearm and the arm. The picture is the result of Co-simulation of step of 2 and 1. We can see both can settle in about 1.5 seconds.

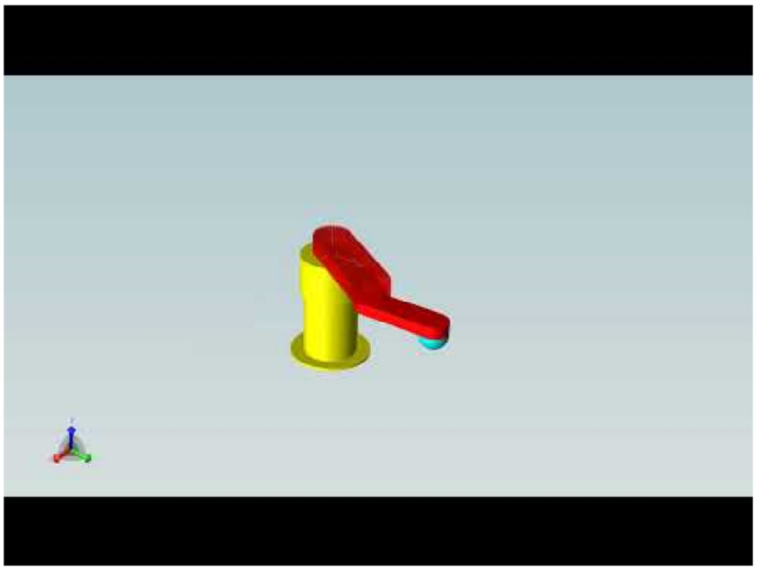
Simulation X cont'd





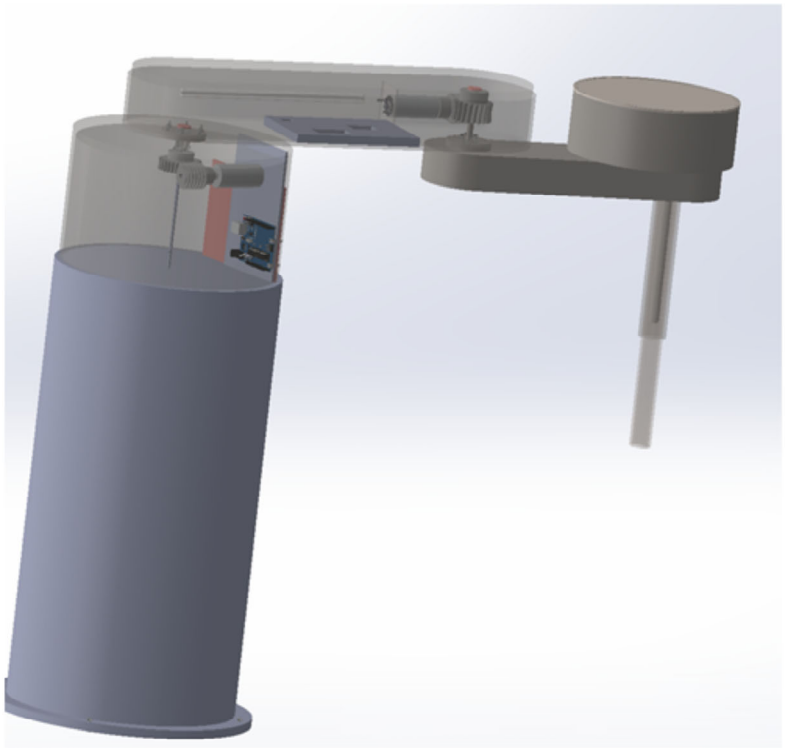
We can use inverse and forward kinematics to give the model x and y coordinate, and get the x and y coordinate of the current location. X axis is along the direction of arm on the model, and y is perpendicular to it. This shows the arm goes to (0.6,0), (0,0.5), (-0.5,0.2) and back, and repeat it a second time. Yellow line is x and blue line is y. We can see that it is fairly fast and relatively smooth, which meets our requirement.

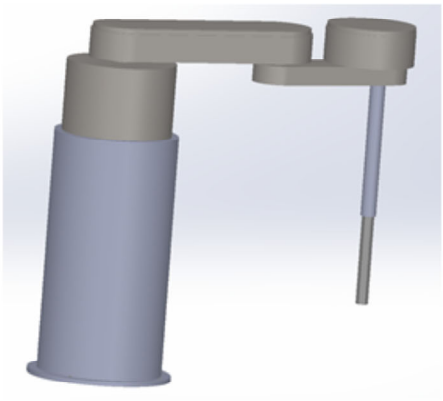
Simulation X cont'd



Then we would like to envision how the arm would be used in real life situation. As the video shows, the arm would pick something up from a conveyor belt, and drop it on the other side.

Extra Feature : Z axis





17

For the extra feature, we achieve one extra degree of freedom.

Our robot arm can move in Z direction.

So, we can pick up things which are at different height.

In order to achieve this extra feature,we used one extra motor and one extra worm gear.

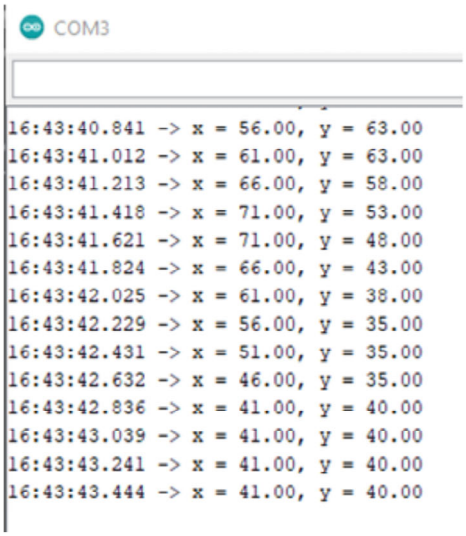
The gear ratio of the worm gear is 30.

1. Motor rotate shaft which is connected to worm gear
1. Worm gear rotates at the same speed with shaft
1. Worm gear is connected to the lead screw through the shaft
1. Worm gear and leads screws rotates at the same speed
1. A hex nut which connected to a rod is used
1. Hex nut cannot rotate because it is surrounded by a shell
1. Rotating Lead screw forces the hex nut to move up or down

The length of the rod which is connected to the hex nut is 75mm.

So the work space is a “cylinder area” with 1.2 meter diameter and 75mm height.

Extra Feature : Joystick control

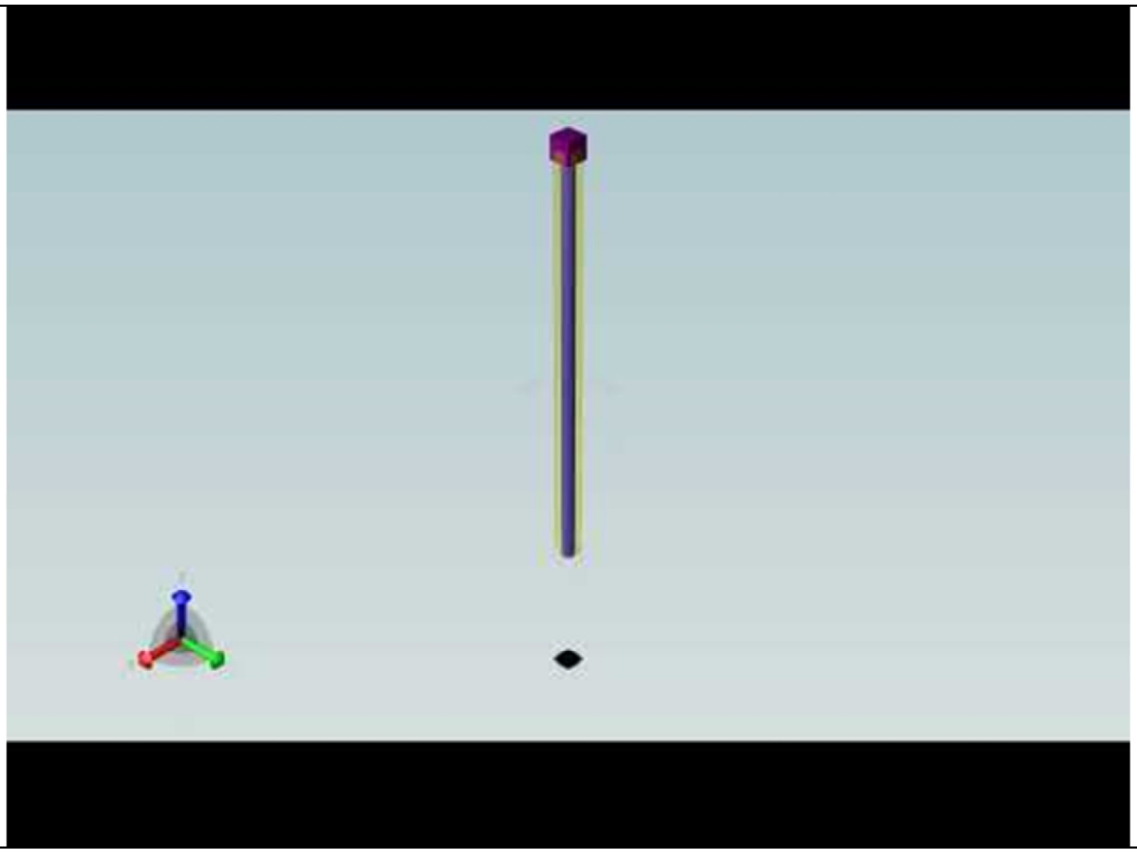



```
void joystick_control(){
    xValue = analogRead(VRX_PIN);
    yValue = analogRead(VRY_PIN);

    if (xcoord <= 100 && xcoord >= 0)
    {
        xcoord = (xValue - 512) / 100 + xcoord;//scale the joystick x reading
    }
    if (ycoord <= 100 && ycoord >= 0)
    {
        ycoord = (yValue - 512) / 100 + ycoord;//scale joystick y reading
    }
    if (ycoord <= 0)
        ycoord = 0;
    if (xcoord <= 0)
        xcoord = 0;

    target_angles = Coord2Angle([ xcoord, ycoord ]);
    //wait for PID to complete movement
}
```


For the next extra feature, we designed a joystick controller for our arm.
It enables user to control the arm manually without the need to input coordinates into the system.
It uses interrupt to read the joystick's analog input (0-1023) on both axis to help determine the direction our arm needs to move.
After we acquired the analog input, this input is scaled down and translated into an x-y coordinate change to our arm's current position.
Then the new target coordinate is fed through our inverse kinematics function to return a target angle for our motors to process during the next interrupt routine.
The left picture shows our debug output for our design, it shows how the arm final coordinate changes in response to the user moving the joystick around.



This is a demo of the control of extra z-axis. We give it a sinusoidal wave and it moves up and down as the rack converts torque to force along z-axis, with good stability.