The Dishwashing robot Manual

Janek de Silva & Adit Roychowdhury

Opportunity

The dishwashing robot opens the doors to a lot more opportunities than a traditional dishwasher. It not only wash dishes faster, with less water and at a higher turnover per minute than a traditional machine, it is also small and fairly portable, able to fit on a countertop or sink side. We wanted to build a robot to keep up with the fast paced lives of college students who don't always have the time to wash dishes, or ample space to do so. Our robot arm cleans multiple dishes in one sitting and requires little loading or maintenance, save for refilling the water and soap. Another benefit is that it scrubs and rinses the dishes as well, further reducing the work needed for a traditional dishwasher.

High level strategy

1. The arm performs a homing sequence and starts with arm raised
2. The plates are placed in the plate holder section which triggers the limit switches
3. The arm moves to the correct position and lowers to begin cleaning.
4. The tool rotates cleaning the plate of large particles
5. The pumps are triggered, for water and soap
6. The arm does a completion rotation, then lifts off to move to the next plate
7. Once all plates are completed the arm begins idling

Desired vs Achieved functionality:

We initially wanted to clean two plates within five minutes, which we achieved in 4 minutes 40 seconds. This however depends on the analog input to cleaning time, which can be adjusted manually to match the level of cleaning required. We also initially planned to clean 3 plates, however we ultimately only had space for 2.

Function Critical design decisions

We had to make numerous design decisions as our project developed and our goals evolved.

1. The gear ratios for each transmission: Tower rotation, arm rotation and tool rotation
2. Forces on the shafts and deciding shaft dimensions and materials, and choosing actuators accordingly
3. Creating a belt tensioning mechanism to allow the pulley to be tensioned from the outside

Shaft 1: Tool Shaft

$F_{reaction}$ = weight of shaft
Symmetrical therefore $F_{net} = W\hat{z}$

$$\tau_{out} = \frac{34}{9}\tau_{in}$$

T_{pulley} is small, negligible force
Force:

$$M_g\hat{z} = 0.2kg, F = 2N$$

Torques:

$$\tau_{in} = 0.23kgcm, \tau_{out} = 0.86kgcm$$

Therefore, negligible radial load

Shaft 2: DC Motor Shaft

T_{pulley} is small, negligible.

τ_{DC} is applied by the motor, no feedback torque from shaft as it is on frictionless bearings

Use of flexible shaft coupler means no significant radial load on motor. Use of 3:1 pulley ratio allows for 3.9kgcm of torque on the tool head.

Shaft 3: Arm servo

$F_b = W$ radial load on shaft

$$M_{eg} = 2N$$

$$\tau_{servo} = L_{arm} \times M_{eg} kgcm = 25 \times 0.2 = 5kgcm$$

We chose a servo that can output more torque at 12kgcm to account for added load from tool motion on plate

Shaft 4: Body servo

No weight on shaft, no torque on shaft due to standoffs and frictionless bearings.

Gear forces

$\tau_{friction} = \mu N$ where N is due to weight of robot

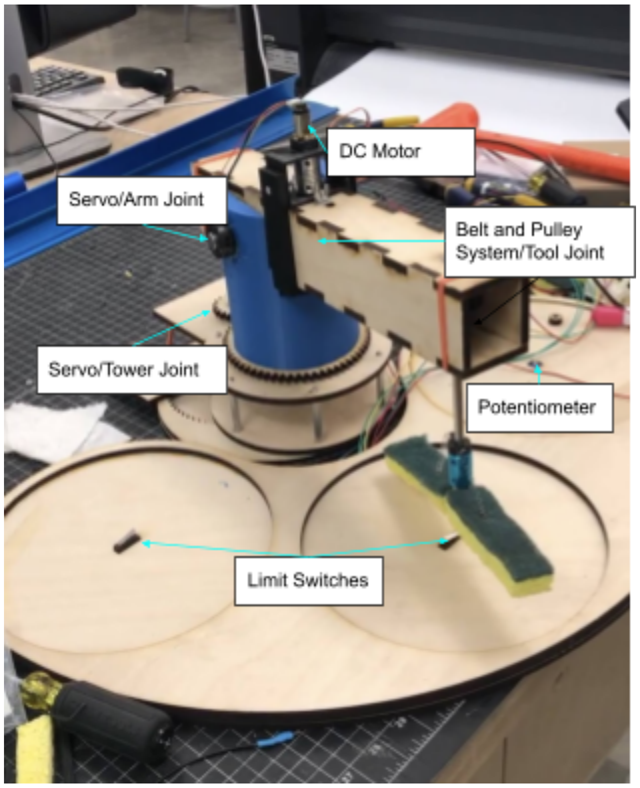
Due to bushing between gear and platform μ is very low

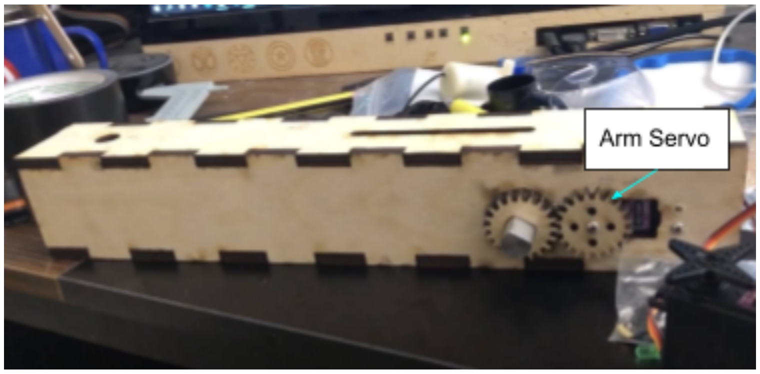
Therefore $\tau_{friction}$ is negligible

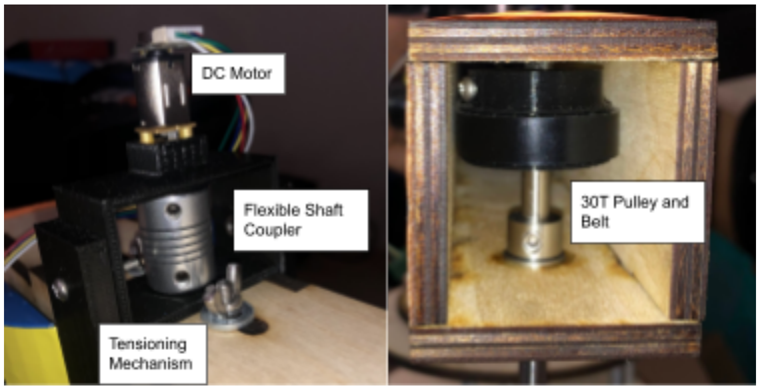
$$\tau_{out} = \frac{N_2}{N_1} \times \tau_{in} = 3\tau_{in}$$

$$\tau_{in} = 12kgcm$$

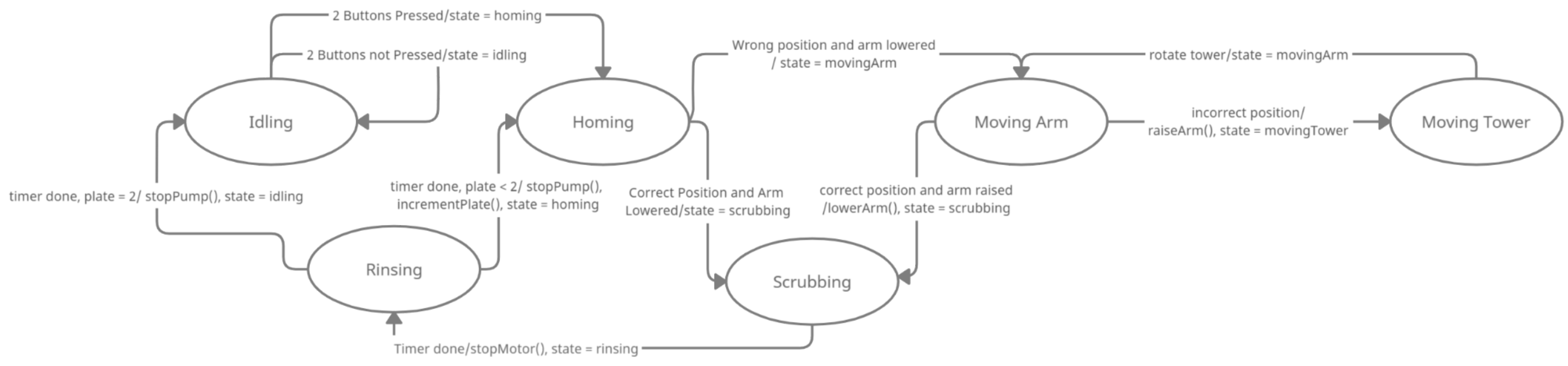
Device

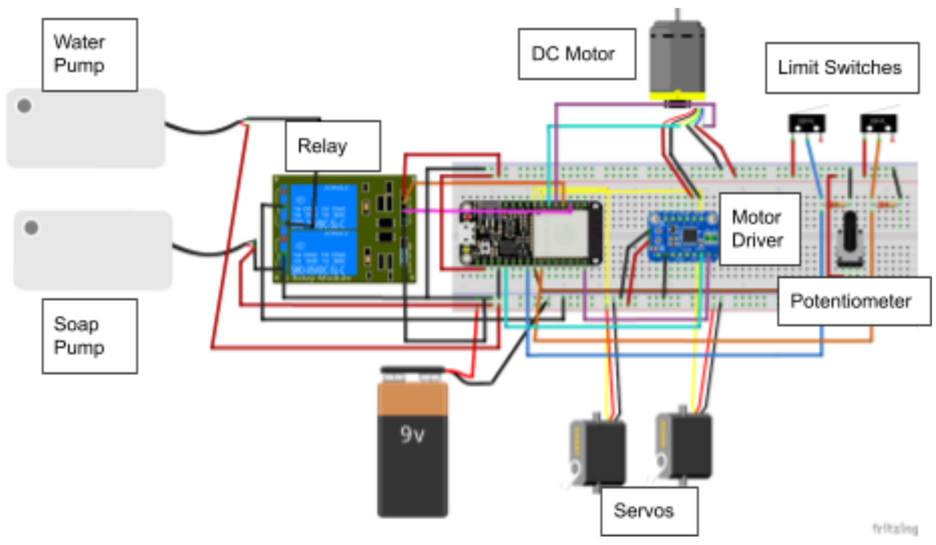






Updated circuit diagram and state transmission diagram


2 Buttons Pressed/state = homing
2 Buttons not Pressed/state = idling
Idling
Homing
Wrong position and arm lowered / state = movingArm
rotate tower/state = movingArm
Moving Arm
incorrect position/ raiseArm(), state = movingTower
Moving Tower
timer done, plate = 2/ stopPump(), state = idling
timer done, plate < 2/ stopPump(), incrementPlate(), state = homing
Correct Position and Arm Lowered/state = scrubbing
correct position and arm raised /lowerArm(), state = scrubbing
Rinsing
Scrubbing
Timer done/stopMotor(), state = rinsing



Reflection

Overall, we were very happy with our outcome in the project. Our transmission systems worked as expected, and we overcame some hurdles in very creative ways. However, our product wasn't perfect and there are a few things we would have done differently. Firstly, the shaft used to connect the motor and pulley was 3mm, and therefore deflected more than expected which made the pulley motion stop sometimes. We would also choose to go with a 360 degree servo for the base motion, so that we can reach more plates and finally, if we had more time and money we would like to build an enclosure to waterproof the system, as we were not able to test our system with real water due to our electronics being exposed.

Appendices

Arduino Code

```
#include <ESP32Servo.h>
#include <ESP32Encoder.h>

ESP32Encoder encoder;
ESP32PWM pwm;

#define tool_motorA1 21
#define tool_motorA2 25

#define ENC1 15
#define ENC2 33
#define BTN_1 4
#define BTN_2 39
#define relayIn1 14
#define relayIn2 32
#define pot 36
#define arm_servoPin 12
#define tower_servoPin 27

Servo arm_servo;
Servo tower_servo;

int minUs = 0;
int maxUs = 2000;
int maxArmTilt = 120;
int numButtonsPressed = 0;
int plate = 0;

unsigned long previousScrubMillis = 0;
unsigned long previousRinseMillis = 0;

long timerLength;

int button[2];
bool armRaised = true;
enum Ste {homing, idling, movingArm, movingTower, scrubbing, rinsing};
int currentTowerServoPos = 0;
int currentArmServoPos = 0;
int omegaSpeed = 0;
int omegaDes = 0;
```

```
float K = 18 / 15;
int D = 255;
int e;
int Kp;
int Ki;
int debounce_delay = 100;
int TotalE;

volatile int count = 0; // encoder count
volatile bool interruptCounter = false;    // check timer interrupt 1
volatile bool deltaT = false;     // check timer interrupt 2
int totalInterrupts = 0;   // counts the number of triggering of the alarm

hw_timer_t * timer0 = NULL;
hw_timer_t * timer1 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

const int freq = 5000;
const int ledChannel_1 = 2;
const int ledChannel_2 = 3;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
const int NOM_PWM_VOLTAGE = 150;

void IRAM_ATTR onTime0() {
  portENTER_CRITICAL_ISR(&timerMux0);
  interruptCounter = true; // the function to be called when timer
interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux0);
}

void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  count = encoder.getCount( );
  encoder.clearCount ( );
  deltaT = true; // the function to be called when timer interrupt is
triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void IRAM_ATTR b1isr() {  // the function to be called when interrupt is
```

```
triggered
  if (digitalRead(BTN_1) == HIGH) {
    b1isrr();
  }
  else {
    b1isrf();
  }

}

void IRAM_ATTR b2isr() {  // the function to be called when interrupt is
triggered
  if (digitalRead(BTN_2) == HIGH) {
    b2isrr();
  }
  else {
    b2isrf();
  }

}

void b1isrr() {  // the function to be called when interrupt is triggered
  //  Serial.println("btn1");
  button[0] = 1;

}

void  b2isrr() {  // the function to be called when interrupt is triggered
  //  Serial.println("btn2");
  button[1] = 1;

}



void  b1isrf() {  // the function to be called when interrupt is triggered
  //Serial.println("btn1o");
  button[0] = 0;

}

void  b2isrf() {  // the function to be called when interrupt is triggered
```

```
  //Serial.println("btn2o");
  button[1] = 0;

}




enum Ste state;

void setup() {
  // put your setup code here, to run once:
  state = idling;
  Serial.begin(9600);
  pinMode(pot, INPUT);
  pinMode(relayIn1, OUTPUT);
  pinMode(relayIn2, OUTPUT);
  timerLength = map(analogRead(pot), 0, 4095, 0, 10000);
  setupButtons();
  setupServos();
  setupEncoder();
  setupMotor();
  setupTimers();
}



void setupServos() {
  // put your setup code here, to run once:
  ESP32PWM::allocateTimer(2);
  ESP32PWM::allocateTimer(3);
  arm_servo.setPeriodHertz(50);      // Standard 50hz servo
  tower_servo.setPeriodHertz(50);
  pinMode(arm_servoPin, OUTPUT);
  pinMode(tower_servoPin, OUTPUT);

  arm_servo.attach(arm_servoPin, minUs, maxUs);
  tower_servo.attach(tower_servoPin, 350, 3000);

  moveToStartArmServo();
  moveToStartTowerServo();
}
```

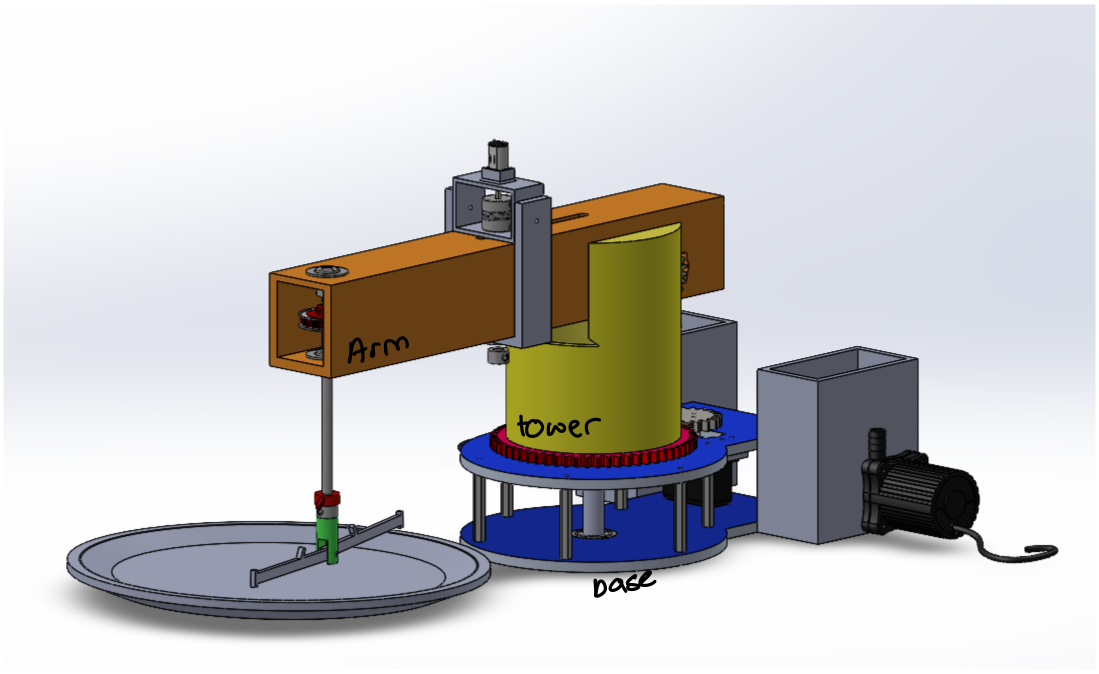
```
void setupButtons() {
  pinMode(BTN_1, INPUT);
  pinMode(BTN_2, INPUT);
  attachInterrupt(BTN_1, b1isr, CHANGE);
  attachInterrupt(BTN_2, b2isr, CHANGE);
}

void setupEncoder() {
  ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull
up resistors
  encoder.attachHalfQuad(ENC1, ENC2); // Attache pins for use as encoder
pins
  encoder.setCount(0);  // set starting count value after attaching

}

void setupMotor() {
  ledcSetup(ledChannel_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(tool_motorA1, ledChannel_1);
  ledcAttachPin(tool_motorA2, ledChannel_2);

  startMotorResponse();
  delay(200);
  stopMotorResponse();
}

void setupTimers() {
  timer0 = timerBegin(0, 80, true);  // timer 0, MWDT clock period = 12.5
ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer0, &onTime0, true); // edge (not level)
triggered
  timerAlarmWrite(timer0, 2000000, true); // 2000000 * 1 us = 2 s,
autoreload true

  timer1 = timerBegin(1, 80, true);  // timer 1, MWDT clock period = 12.5
ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer1, &onTime1, true); // edge (not level)
triggered
```

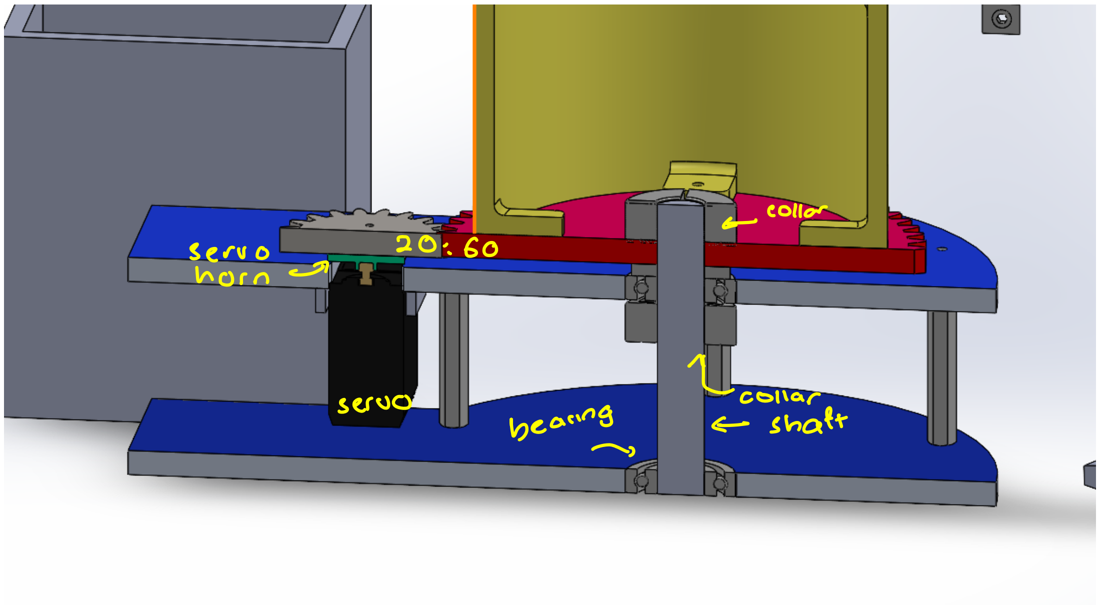
```
  timerAlarmWrite(timer1, 10000, true); // 10000 * 1 us = 10 ms, autoreload
true

  // at least enable the timer alarms
  timerAlarmEnable(timer0); // enable
  timerAlarmEnable(timer1); // enable

}

void moveToStartArmServo() {
  raiseArm();
  //lowerArm();


}

void moveToStartTowerServo() {
  int pos = 0;
  for (pos = 0; pos <= 5; pos += 1) { // goes from 180 degrees to 0 degrees
5
    tower_servo.write(pos);    // tell servo to go to position in variable
'pos'
    delay(50);              // waits 15ms for the servo to reach the
position
  }

  for (pos = 5; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
5
    tower_servo.write(pos);    // tell servo to go to position in variable
'pos'
    delay(50);              // waits 15ms for the servo to reach the
position
  }
}

int buttonEventChecker() {
  if (button[0] && button[1]) {
    plate = 0;
    state = homing;
  }
  else {
    state = idling;
  }
}

void incrementPlate() {
  plate = plate + 1;
  if (plate > 1) {
    Serial.println("FINISHED CLEANING");
    delay(10000);
    state = idling;

  }
}


int correctPosition() {
  if (plate == 0) {
    return 0;
  }
  else if (plate == 1) {
    return 180;
  }
}

void stopMotorResponse() {
  Serial.println("stop motor");
  ledcWrite(ledChannel_2, LOW);
  ledcWrite(ledChannel_1, LOW);

}

void startMotorResponse() {
  Serial.println("start motor");
  ledcWrite(ledChannel_1, D);
  ledcWrite(ledChannel_2, LOW);
}

bool lowerArm() {
  Serial.println("Raise arm to 0 degrees");
  int pos = 0;
  for (pos = 0; pos <= 90; pos += 1) { // goes from 180 degrees to 0
degrees
    arm_servo.write(pos);    // tell servo to go to position in variable
```

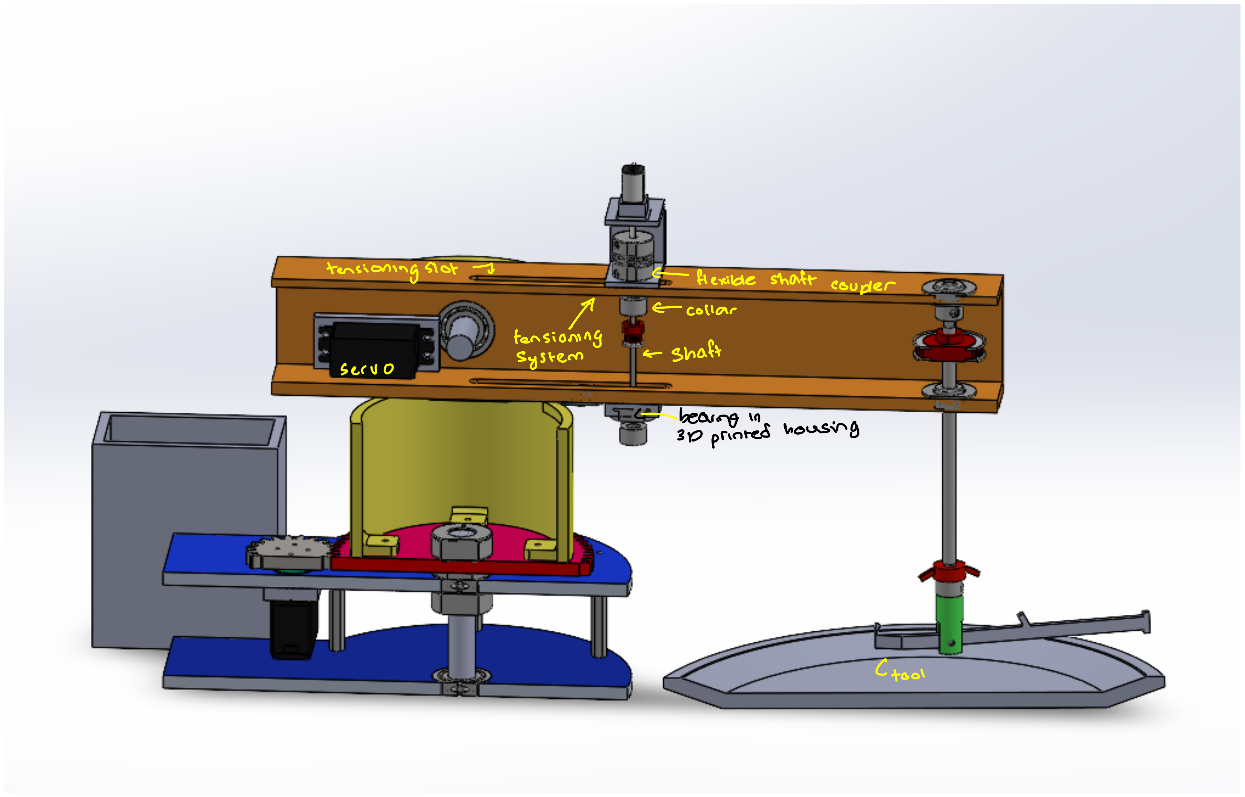
```
'pos'
    delay(50);              // waits 15ms for the servo to reach the
position
  }
}

bool raiseArm() {
  Serial.println("raise arm to 90 degrees");
  int pos = 90;
  for (pos = 90; pos >= 0; pos -= 1) { // goes from 180 degrees to 0
degrees
    arm_servo.write(pos);    // tell servo to go to position in variable
'pos'
    delay(20);              // waits 15ms for the servo to reach the
position
  }
}

void startRinse() {
  Serial.println("starting pump");
  digitalWrite(relayIn1, HIGH);
}

void stopRinse() {
  Serial.println("stopping pump");
  digitalWrite(relayIn1, LOW);
}
bool rotateTower(int correctPosition, int currentPosition) {
  Serial.print("Rotating tower to ");
  Serial.print(correctPosition);
  Serial.print(" degrees");
  Serial.println();
  if (currentPosition < correctPosition) {
    int pos = currentPosition;
    for (pos = currentPosition; pos <= correctPosition; pos += 1) { // goes
from 180 degrees to 0 degrees
      tower_servo.write(pos);    // tell servo to go to position in
variable 'pos'
      delay(50);              // waits 15ms for the servo to reach the
position
    }
  } else if (currentPosition > correctPosition) {
    int pos = currentPosition;
    for (pos = currentPosition; pos >= correctPosition; pos -= 1) { // goes
from 180 degrees to 0 degrees
      tower_servo.write(pos);    // tell servo to go to position in
variable 'pos'
      delay(50);             // waits 15ms for the servo to reach the
position
    }
  }

}

void rotateTool() {
  startMotorResponse();
  if (deltaT) {
    portENTER_CRITICAL(&timerMux1);
    deltaT = false;
    portEXIT_CRITICAL(&timerMux1);

    omegaSpeed = count;
    omegaDes = 1;
    int start_time = millis();
    int end_time = timerLength;
    int run_time = 0;
    while (run_time <= end_time) {
      if (omegaSpeed != omegaDes) {
        if (TotalE <= 18 ) {
          e = (omegaDes - omegaSpeed);
          TotalE = TotalE + e;
        }

        D = (Kp * e) + (Ki * TotalE / 10 );
      }
      else
      {
        TotalE = 0;
      }

      if (D > MAX_PWM_VOLTAGE) {
        D = MAX_PWM_VOLTAGE;
      }
      else if (D < -MAX_PWM_VOLTAGE) {
        D = -MAX_PWM_VOLTAGE;
      }
      run_time = millis() - start_time;
    }


  }
  deltaT = true;
  omegaSpeed = 0;
  TotalE = 0;
  D = 255;
}

void printButtons() {
  for (int i = 0; i < 2; i++) {
    Serial.print(button[i]);
  }
  Serial.println();

}
void loop() {
  // put your main code here, to run repeatedly:
  int currentTowerPos = tower_servo.read();
  int currentArmPos = arm_servo.read();
  printButtons();
  delay(100);
  switch (state) {
    case idling :
      Serial.println("Idling");
      buttonEventChecker();
      break;
    case homing:
      Serial.println("Homing");
      if ((currentTowerPos == correctPosition()) && !(armRaised)) {
        state = scrubbing;
      }
      else {
        state = movingArm;
      }
      break;
    case movingArm:
      Serial.println("MovingArm");
      if (!(currentTowerPos == correctPosition()) && armRaised) {
        state = movingTower;
      }
      else if (!(currentTowerPos == correctPosition()) && !armRaised) {
        raiseArm();
        armRaised = true;
        state = movingTower;
      }
      else if ((currentTowerPos == correctPosition()) && armRaised) {
        lowerArm();
        armRaised = false;
        state = scrubbing;
      }

      break;
    case movingTower:
      Serial.println("movingTower");
      rotateTower(correctPosition(), currentTowerPos);
      state = movingArm;
      break;
    case scrubbing:
      Serial.println("scrubbing");
      Serial.println(timerLength);
      rotateTool();

      delay(timerLength);
      stopMotorResponse();
      state = rinsing;
      break;
    case rinsing:
      Serial.println("rinsing");
      startRinse();
      Serial.println(timerLength);
      delay(timerLength);
      stopRinse();
      incrementPlate();
      Serial.println("plate");
      Serial.println(plate);
      if (plate < 2) {
        state = homing;
      } else {
        state = idling;
```

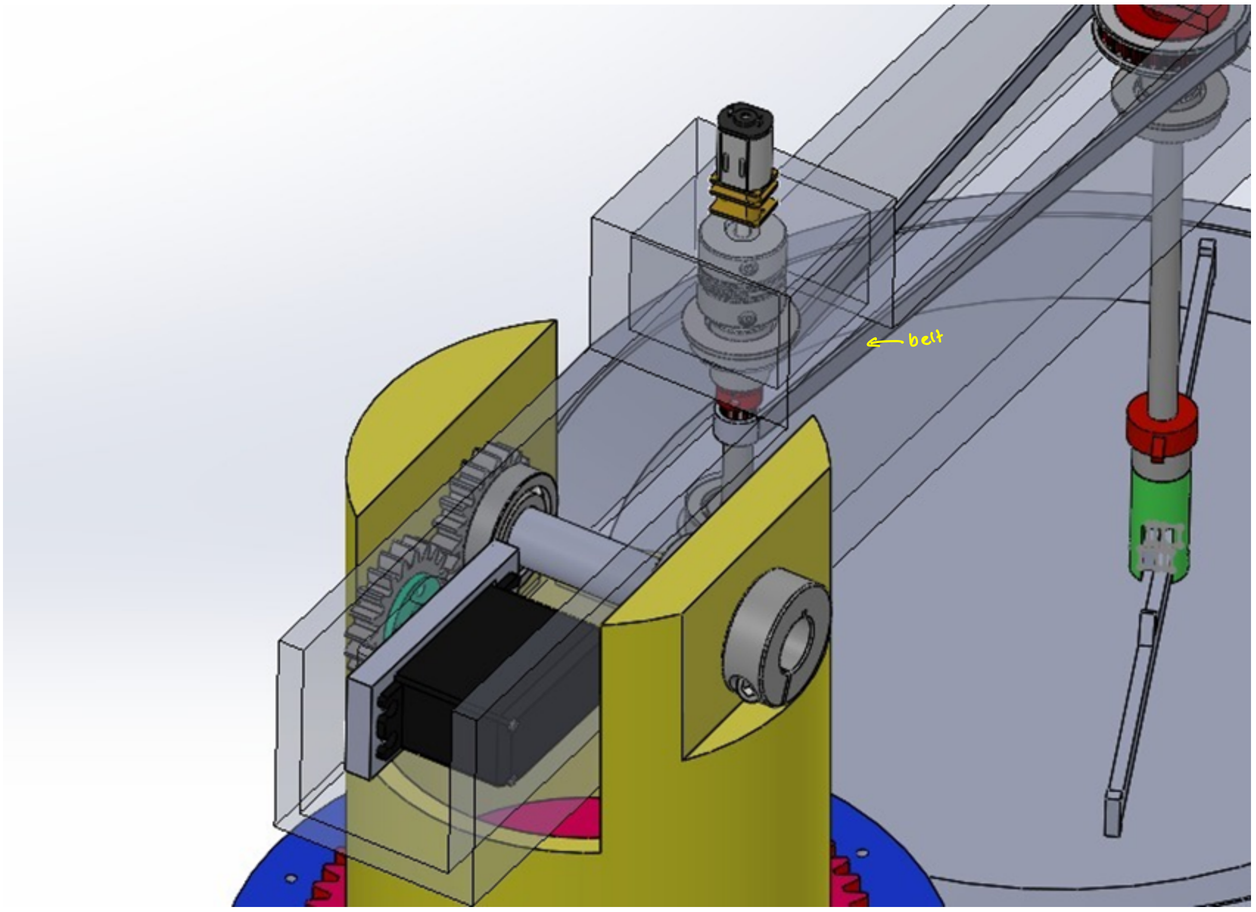
```
            }
            break;

    }
}
```

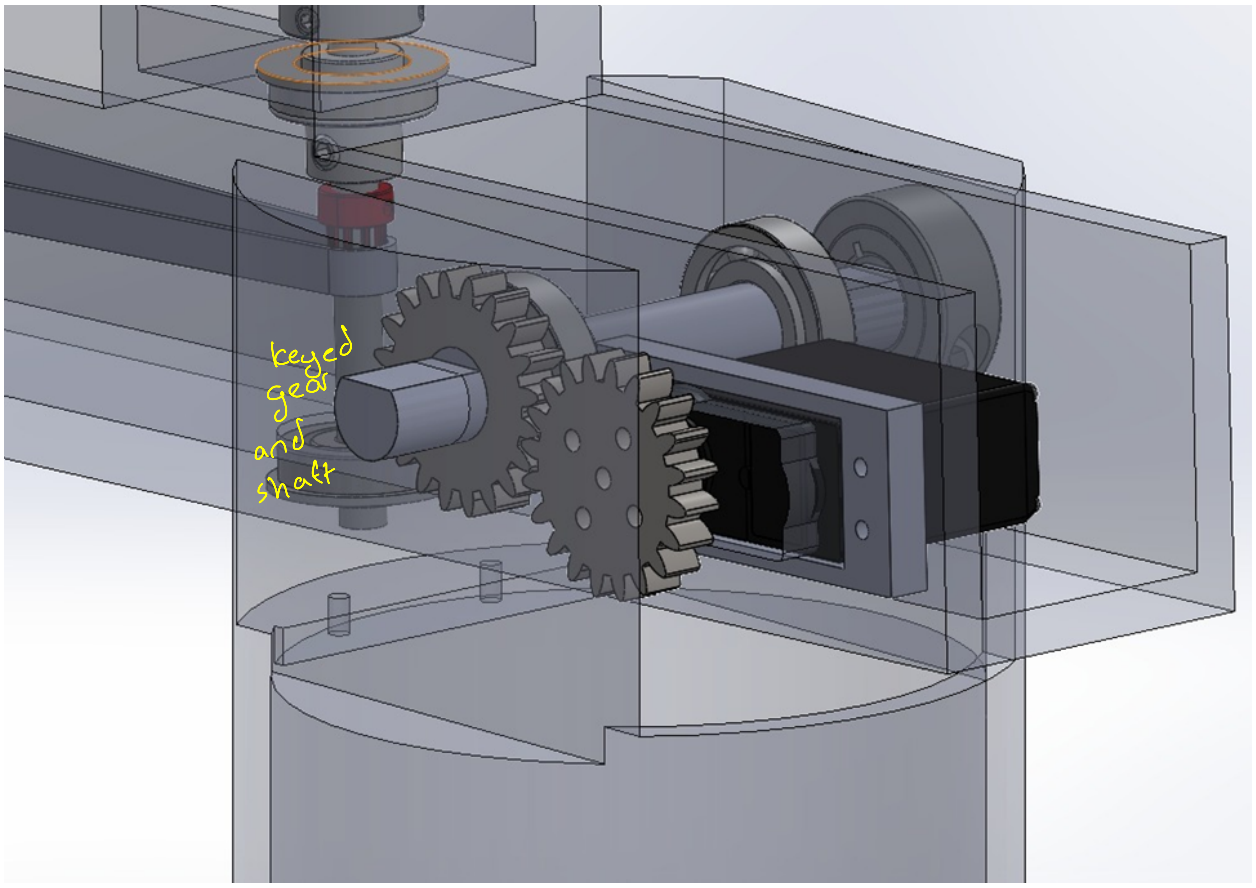
CAD Model

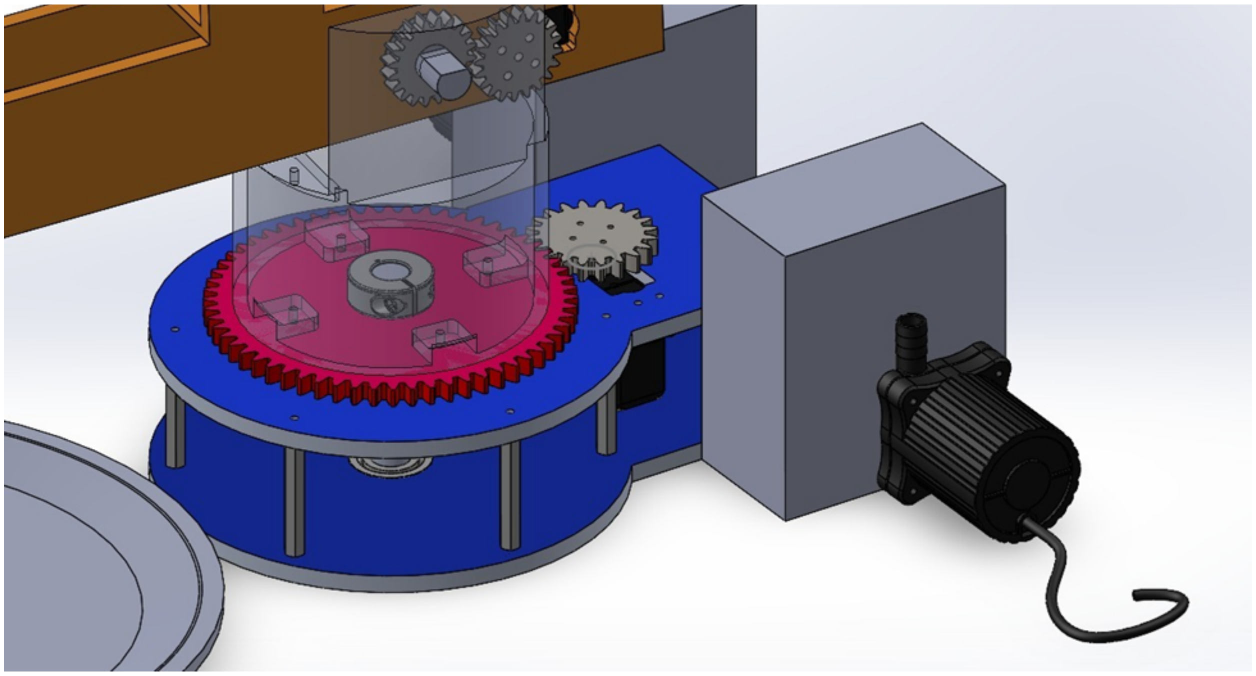

Arm
tower
base

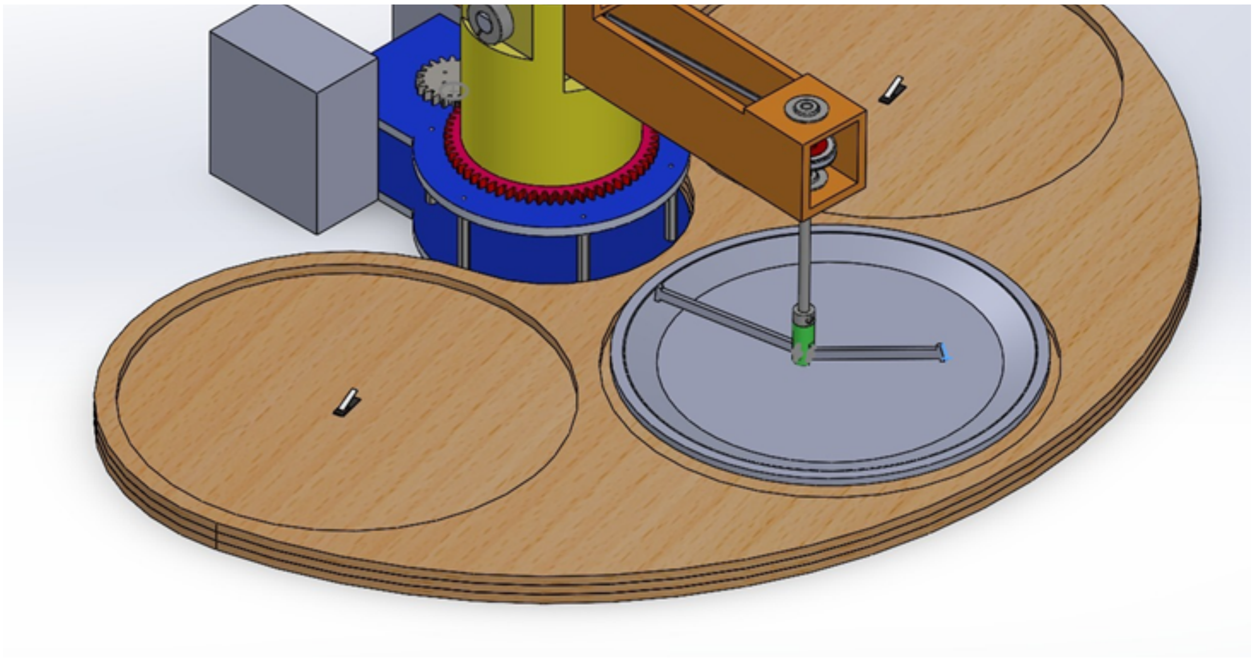

servo horn
20 : 60
collar
servo
bearing
collar
shaft


tensioning slot
flexible shaft coupler
collar
servo
tensioning system
shaft
bearing in 3D printed housing
tool

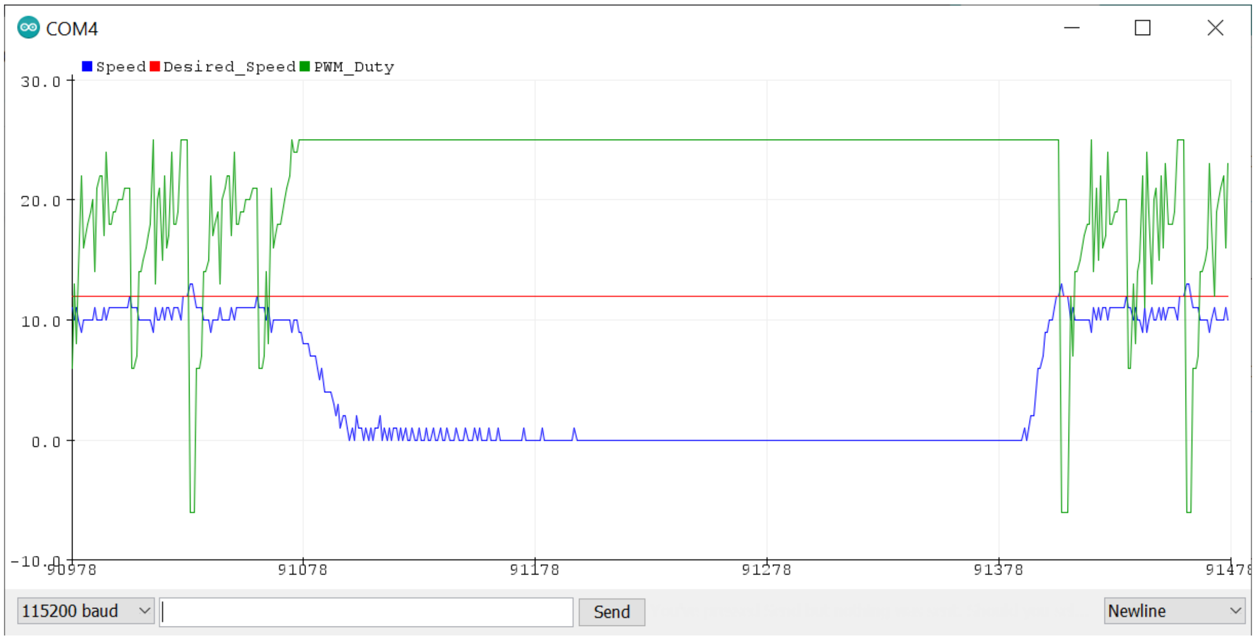

belt


keyed gear and shaft





Graph showing PI control on DC Motor


COM4
🟦 Speed 🟥 Desired_Speed 🟩 PWM_Duty
115200 baud
Send
Newline