Defining the Node.

Friday, December 29, 2023 3:21 PM

```
ros::NodeHandle_<ArduinoHardware,
                 MAX_PUBLISHERS,
                 MAX_SUBSCRIBERS,
                 PUBLISHERS_BUFFER_SIZE,
                 SUBSCRIBERS_BUFFER_SIZE>
  nh;
```

Defining publishers and subscribers

Friday, December 29, 2023 3:50 PM

defining object type "float 32"

```
std_msgs::Float32MultiArray pwm_feedback_msg;
ros::Publisher pwm_pub(TOPIC_NAME_PWM_FEEDBACK, &pwm_feedback_msg);   //pwm topic

std_msgs::Float32MultiArray velocity_feedback_msg;
ros::Publisher velocity_pub(TOPIC_NAME_VELOCITY_FEEDBACK, &velocity_feedback_msg);
```

defining Publisher. "PWM_Pub"

topic

object Published.

```
ros::Subscriber<std_msgs::Float32MultiArray> sub_speeds(
    TOPIC_NAME_SETPOINTS,
    &callback_speeds);  // Subscriber to subscribe from Master
```

```
ros::Subscriber<std_msgs::Float32MultiArray> params("parameters", &callback_parameters);   //parameters topic

//controlled velocity(0) topic
std_msgs::Float64 vel_0;
ros::Publisher encoders_0("encoders_0", &vel_0);
//controlled velocity(1) topic
std_msgs::Float64 vel_1;
ros::Publisher encoders_1("encoders_1", &vel_1);

//encoders _counts

std_msgs::Int16 c_0msg;
ros::Publisher c_0("c_0", &c_0msg);

std_msgs::Int16 c_1msg;
ros::Publisher c_1("c_1", &c_1msg);
```

what Parameters??

Publish velocities.

counts ??

Defining callback functions

Friday, December 29, 2023 3:53 PM

```
// Function to call Speeds from output matrics data from master
void callback_speeds(const std_msgs::Float32MultiArray &speeds_msg)
{
  double speed_0=(double)speeds_msg.data[0];
  double speed_1=(double)speeds_msg.data[1];
  if(speed_0>MAX_SPEED)
    speed_0=MAX_SPEED;
  else if (speed_0<-MAX_SPEED)
    speed_0=-MAX_SPEED;
  if(speed_1>MAX_SPEED)
    speed_1=MAX_SPEED;
  else if (speed_1<-MAX_SPEED)
    speed_1=-MAX_SPEED;
  velocity_pid[0].setpoint(speed_0);  // Get setpoint from output matrix in master
  velocity_pid[1].setpoint(speed_1);  // Get setpoint from output matrix in master
}
```

Recieved object "array"

assign array elements to local variables.

ensure speed is within "limits"

```
void callback_parameters(const std_msgs:: Float32MultiArray& parameters)    //parameters method
{
  velocity_pid[0].tune(parameters.data[0],parameters.data[1],parameters.data[2]);
  velocity_pid[1].tune(parameters.data[3],parameters.data[4],parameters.data[5]);
}
```

tuning Pid Parameters

"SetuP"

Friday, December 29, 2023 4:02 PM

```
void setup()
{
 // SERIAL_ROS.begin(BAUD_RATE);
  Serial1.begin(BAUD_RATE);

  // PWM setup
  analogWriteFrequency(FREQUENCY);
  analogWriteResolution(16);

  initRos();
  initEncoders();
  initMotorDrivers();
  initPid();

  //  Start scheduler and LED indication
  pinMode(PC13, OUTPUT);
  digitalWrite(PC13, LOW);
  //digitalWrite(LED_BUILTIN, HIGH);

  myTimer.every(PID_PERIOD, velocityPidRoutine);
}
```

starting Serial Monitor.

Initialization functions.

For sample time "PID"

"loop"

Friday, December 29, 2023 4:04 PM

```
void loop()
{
  myTimer.update();
  nh.spinOnce();
}

// ================================================================
// ===                    CLOSED LOOP CONTROL                   ===
// ================================================================
```

"PID_Routine

Friday, December 29, 2023 4:09 PM

```
void velocityPidRoutine(void)
{
    // main routine, measures velocity, passes velocity to the PID then applies
    // the output to the motors

    // Motor1
    speeds[0] = (((1.0*(counter[0]-lastCounter[0])/ENCODER_RESOLUTION)*2*PI))/(1.0*(PID_PERIOD/1000.0));   //speed(0) (ra
    pwm[0]       = velocity_pid[0].calculate(speeds[0]);

    if(velocity_pid[0].setpoint()==0){
      velocity_pid[0].integralAccumulator = 0;
    }

    if (velocity_pid[0].setpoint()==0 && abs(velocity_pid[0].setpoint()-speeds[0])<=DEADZONE)
        pwm[0]=0;
    else if (pwm[0] > 0)
        digitalWrite(PIN_MOTOR_DIR[0], HIGH);
    else if (pwm[0] < 0)
        digitalWrite(PIN_MOTOR_DIR[0], LOW);
    analogWrite(PIN_MOTOR_PWM[0], abs(pwm[0]));
    lastCounter[0] = counter[0];
```

Measure speed from encoder count.

Calculate PWM using PID

If at set point or at "dead_zone" don't move.

Direction Determination

Write PWM

Update Counter.

Repeat for Motor 2

```
    // Motor2
    speeds[1] = (((1.0*(counter[1]-lastCounter[1])/ENCODER_RESOLUTION)*2*PI))/(1.0*(PID_PERIOD/1000.0));   //speed(1) (radian/s)
    pwm[1]       = velocity_pid[1].calculate(speeds[1]);

    if(velocity_pid[1].setpoint()==0){
      velocity_pid[1].integralAccumulator = 0;
    }

    if (velocity_pid[1].setpoint()==0 && abs(velocity_pid[1].setpoint()-speeds[1])<=DEADZONE)
        pwm[1]=0;
    else if (pwm[1] > 0)
        digitalWrite(PIN_MOTOR_DIR[1], HIGH);
    else if (pwm[1] < 0)
        digitalWrite(PIN_MOTOR_DIR[1], LOW);

    analogWrite(PIN_MOTOR_PWM[1], abs(pwm[1]));
    lastCounter[1] = counter[1];
```

Pid_routine

Friday, December 29, 2023 4:19 PM

```
  vel_0.data=(double)speeds[0];
  vel_1.data=(double)speeds[1];

  encoders_0.publish(&vel_0);
  encoders_1.publish(&vel_1);

//PUBLISH COUNTS
  c_0msg.data=(double)counter[0];
  c_1msg.data=(double)counter[1];

  c_0.publish(&c_0msg);
  c_1.publish(&c_1msg);
```

↳ update current velocity.

↳ Publish to "encoder"

↳ update counts

↳ Publish to "c_0" "c_1"

↳ for debugging purposes.

```
  // Debugging
  debugROS();
/   debugSERIAL();
```

↳ Debugging function.

"Encoder Service Routines"

Friday, December 29, 2023 4:24 PM

```
void encoderISR_A0(void)
{
  counter[0] += digitalRead(PIN_ENCODER_A[0]) == digitalRead(PIN_ENCODER_B[0]) ? -1 : 1;
}
void encoderISR_B0(void)
{
  counter[0] += digitalRead(PIN_ENCODER_A[0]) != digitalRead(PIN_ENCODER_B[0]) ? -1 : 1;
}

void encoderISR_A1(void)
{
  counter[1] += digitalRead(PIN_ENCODER_A[1]) != digitalRead(PIN_ENCODER_B[1]) ? -1 : 1;
}
void encoderISR_B1(void)
{
  counter[1] += digitalRead(PIN_ENCODER_A[1]) == digitalRead(PIN_ENCODER_B[1]) ? -1 : 1;
}
```

↑ ternary operator.

↑ if false.

↑ if true

for counter zero

(A0) If A0 == B0 Co++

else Co--

(B0) If A0 != B0 Co++

else Co--

Initialization functions

Friday, December 29, 2023 4:34 PM

```
void initRos()
{
    //(nh.getHardware())->setPort(&Serial1);
    //(nh.getHardware())->setBaud(BAUD_RATE);

//    (nh.getHardware())->setPort(&Serial1);
//    (nh.getHardware())->setBaud(BAUD_RATE);
    nh.initNode();
    nh.advertise(pwm_pub);
    nh.advertise(velocity_pub);
    nh.advertise(encoders_0);
    nh.advertise(encoders_1);
    nh.advertise(c_0);
    nh.advertise(c_1);
    nh.subscribe(sub_speeds);
    nh.subscribe(params);

    // ROS Float32MultiArray msg setup
    char dim0_label[] = "PWM";
    pwm_feedback_msg.layout.dim = (std_msgs::MultiArrayDimension *) malloc(sizeof(std_msgs::MultiArrayDimension) * 2);
    pwm_feedback_msg.layout.dim[0].label = dim0_label;
    pwm_feedback_msg.layout.dim[0].size = 2;
    pwm_feedback_msg.layout.dim[0].stride = 1*2;
    pwm_feedback_msg.data = (float *)malloc(sizeof(float)*2);
    pwm_feedback_msg.layout.dim_length = 0;
    pwm_feedback_msg.data_length = 2;
```

nh → node class

Initialize node, advertise variables, subscribe to topics

allocating space of (2) dimensions

PWM array setup

```
char dim1_label[] = "Velocity";
velocity_feedback_msg.layout.dim = (std_msgs::MultiArrayDimension *) malloc(sizeof(std_msgs::MultiArrayDimension) * 2);
velocity_feedback_msg.layout.dim[0].label = dim1_label;
velocity_feedback_msg.layout.dim[0].size = 2;
velocity_feedback_msg.layout.dim[0].stride = 1*2;
velocity_feedback_msg.data = (float *)malloc(sizeof(float)*2);
velocity_feedback_msg.layout.dim_length = 0;
velocity_feedback_msg.data_length = 2;
```

divided

same for Velocity.

Initialization functions

Friday, December 29, 2023 4:41 PM

```
void initEncoders()
{
    // Motor1 Encoder
    pinMode(PIN_ENCODER_A[0], INPUT_PULLUP);
    pinMode(PIN_ENCODER_B[0], INPUT_PULLUP);
    attachInterrupt(PIN_ENCODER_A[0], encoderISR_A0, CHANGE);
    attachInterrupt(PIN_ENCODER_B[0], encoderISR_B0, CHANGE);

    // Motor2 Encoder
    pinMode(PIN_ENCODER_A[1], INPUT_PULLUP);
    pinMode(PIN_ENCODER_B[1], INPUT_PULLUP);
    attachInterrupt(PIN_ENCODER_A[1], encoderISR_A1, CHANGE);
    attachInterrupt(PIN_ENCODER_B[1], encoderISR_B1, CHANGE);
}

void initMotorDrivers()
{
    // Motor1
    pinMode(PIN_MOTOR_DIR[0], OUTPUT);
    pinMode(PIN_MOTOR_PWM[0], OUTPUT);
    digitalWrite(PIN_MOTOR_DIR[0], LOW);
    digitalWrite(PIN_MOTOR_PWM[0], LOW);

    // Motor2
    pinMode(PIN_MOTOR_DIR[1], OUTPUT);
    pinMode(PIN_MOTOR_PWM[1], OUTPUT);
    digitalWrite(PIN_MOTOR_DIR[1], LOW);
    digitalWrite(PIN_MOTOR_PWM[1], LOW);
}
```

assign Pins

attach interrupts to Pins → ISR

assign Pins and set to low

```
void initPid()
{
    // Motor1
    velocity_pid[0].setpoint(sp[0]);
    velocity_pid[0].limitOutput(MIN_OUTPUT, MAX_OUTPUT);

    // Motor2
    velocity_pid[1].setpoint(sp[1]);
    velocity_pid[1].limitOutput(MIN_OUTPUT, MAX_OUTPUT);
}
```

→ Initializing PID

Simple unit conversions

Friday, December 29, 2023 4:44 PM

```
double rps2ppms(double rps)  // Revelution per second to pulse per millisecond
{
  return (rps * ENCODER_RESOLUTION) / (1000 * 60 * 60);
}

double ppms2cmps(double pulses_per_ms)  // pulse per millisecond to cm per second
{
  return (pulses_per_ms * 1000.0 / ENCODER_RESOLUTION) * (2 * PI * RADIUS);
}

double ppms2rps(double pulses_per_ms)  // pulse per millisecond to cm per second
{
  return (pulses_per_ms * 1000.0 / ENCODER_RESOLUTION) * (60 * 60);
}

double cmps2ppms(double cm_per_s)  // cm per second to pulse per millisecond
{
  return (cm_per_s / (2 * PI * RADIUS) * ENCODER_RESOLUTION) / 1000;
}
```

Debugging function

Friday, December 29, 2023 4:45 PM

```
void debugROS()
{
    // Feedback from PID output
    pwm_feedback_msg.data[0] = (float)pwm[0];
    pwm_feedback_msg.data[1] = (float)pwm[1];
    pwm_pub.publish(&pwm_feedback_msg);

    // Feedback from encoder
    velocity_feedback_msg.data[0] = (float)ppms2cmps(speeds[0]);     // Measured velocity in RPS
    velocity_feedback_msg.data[1] = (float)ppms2cmps(speeds[1]);     // Measured velocity in RPS
    velocity_pub.publish(&velocity_feedback_msg);
}
```