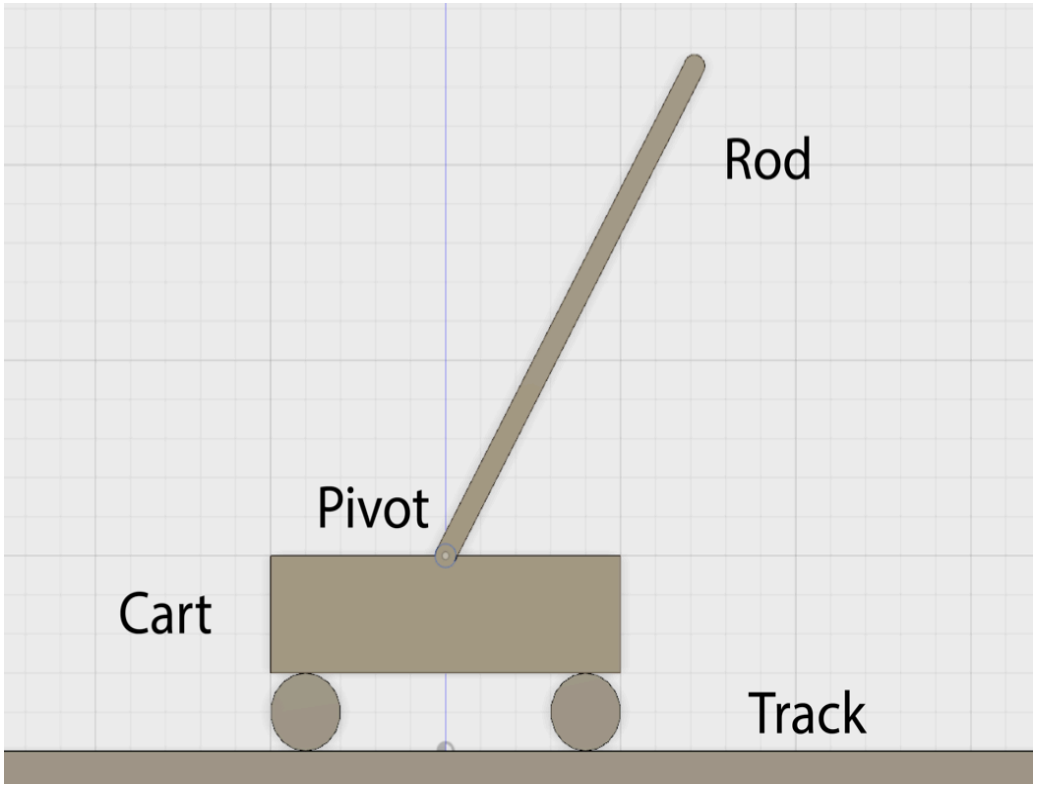
ROCO218: Control Engineering
Dr Ian Howard

Lecture 8

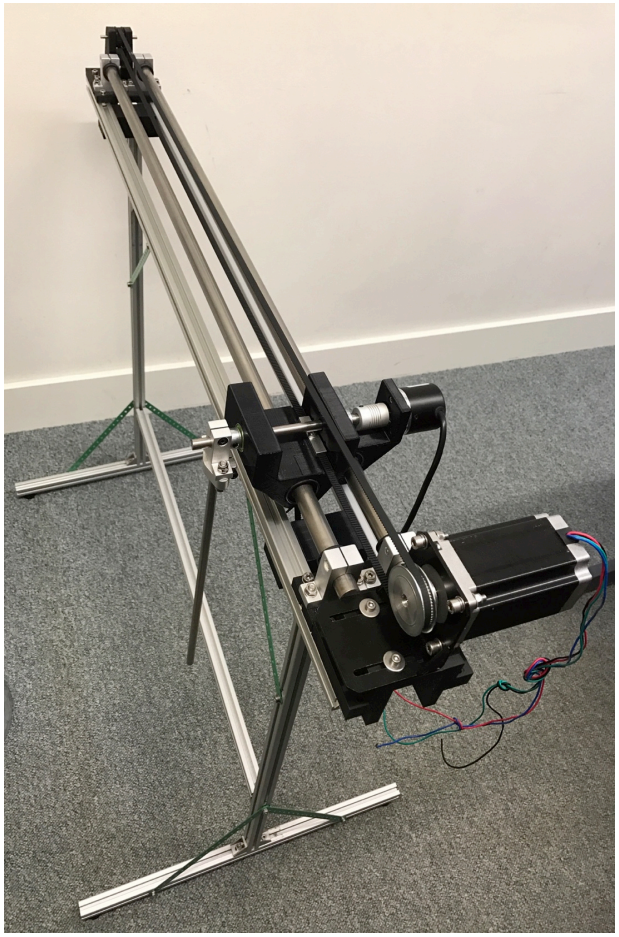
Real inverted pendulum practical

Inverted pendulum main components

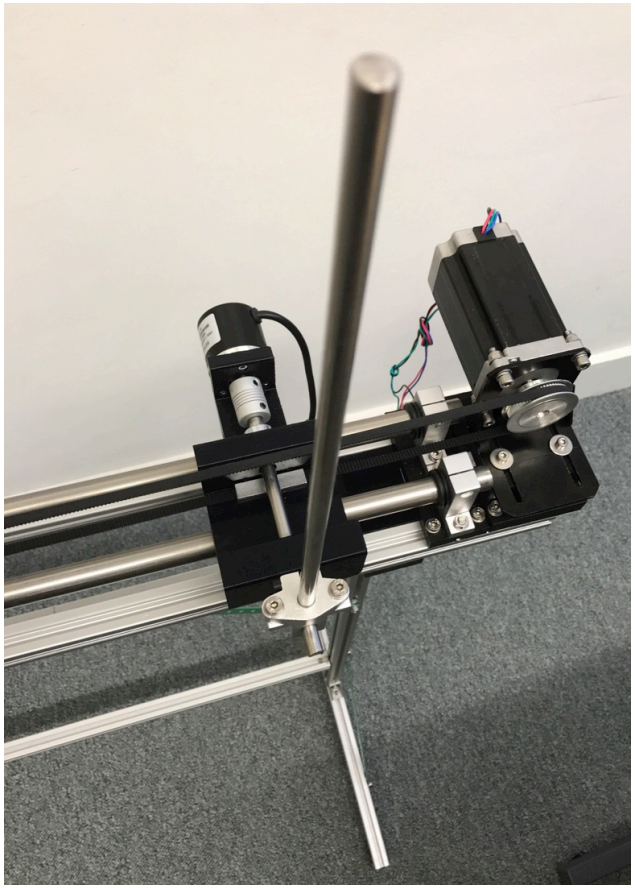


- By moving the cart to the left and the right it is possible to balance the pole and keep it upright

Pendulum system mounted on its aluminum profile stand

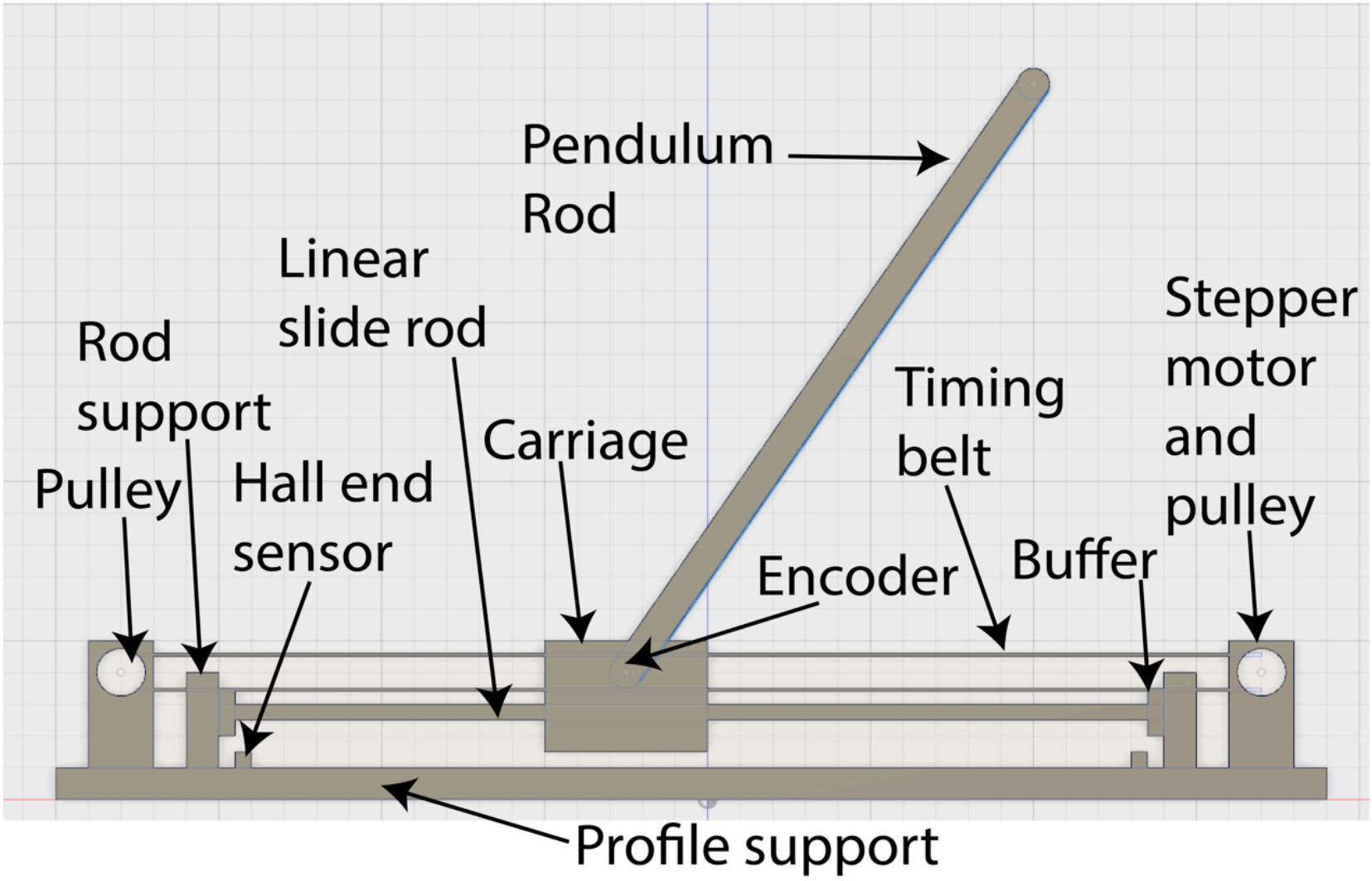


Stand allows the pendulum to rotate freely and avoid collisions.

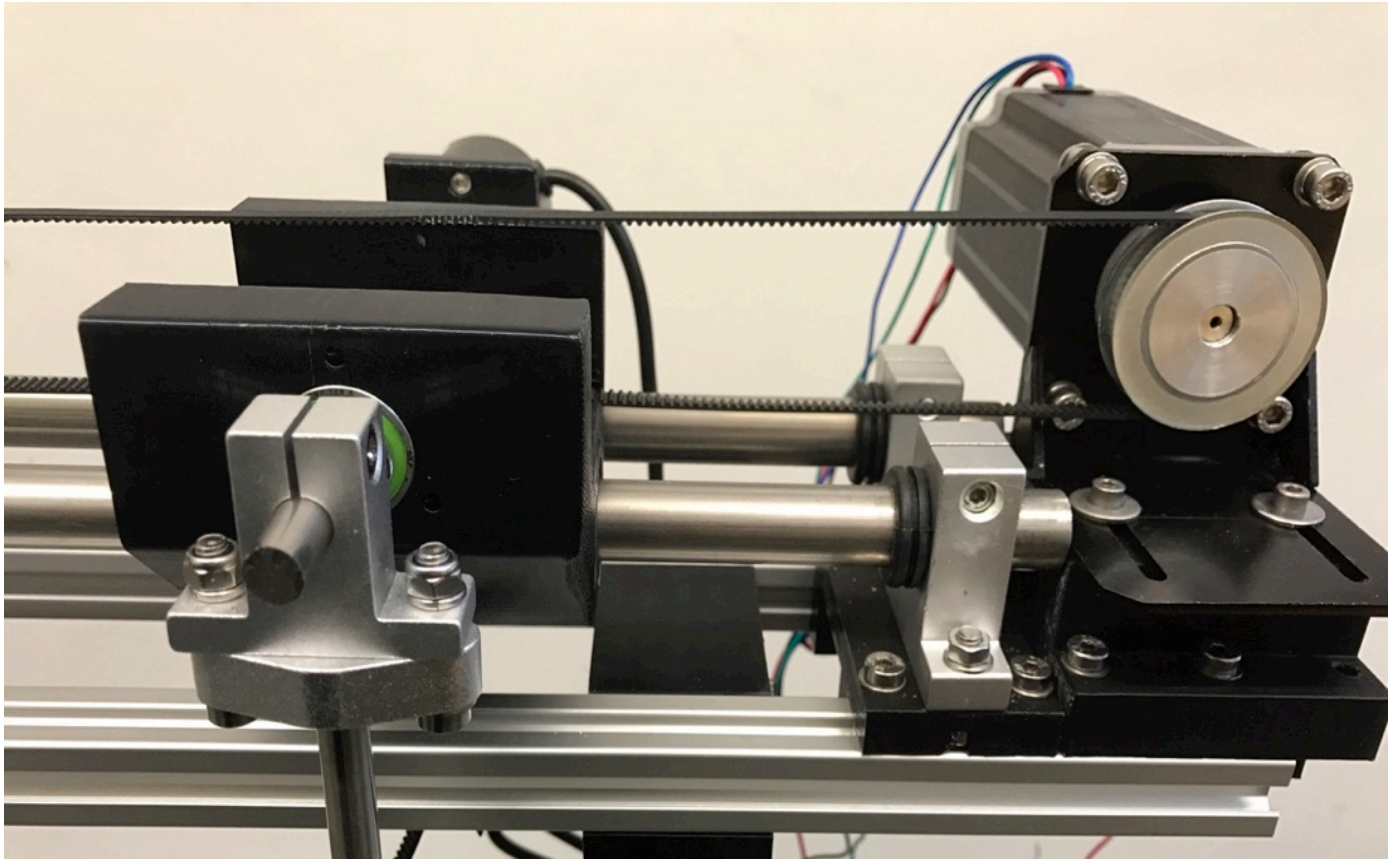


Pendulum system with the rod shown vertically in its unstable position

Inverted pendulum main components

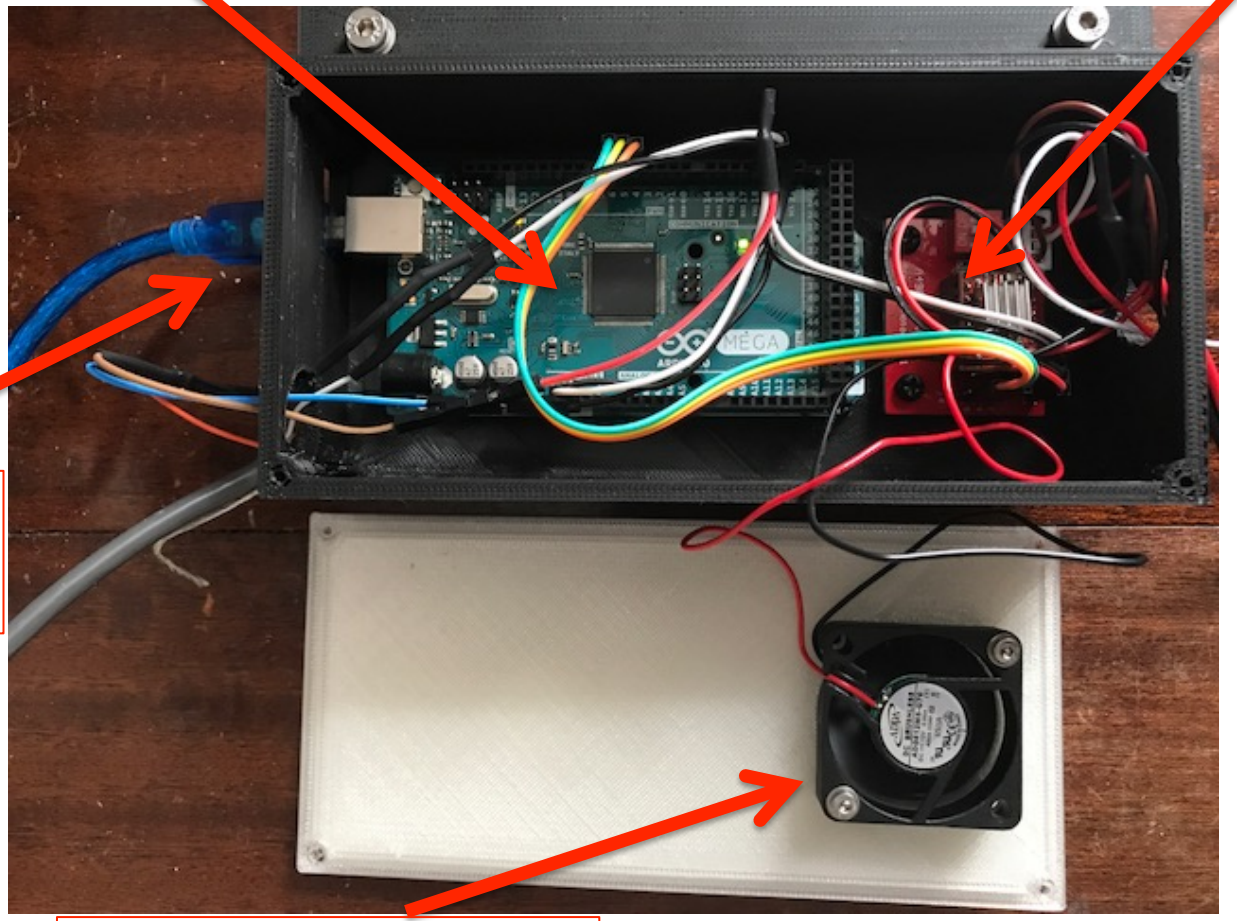


Pendulum drive motor and carriage mechanism

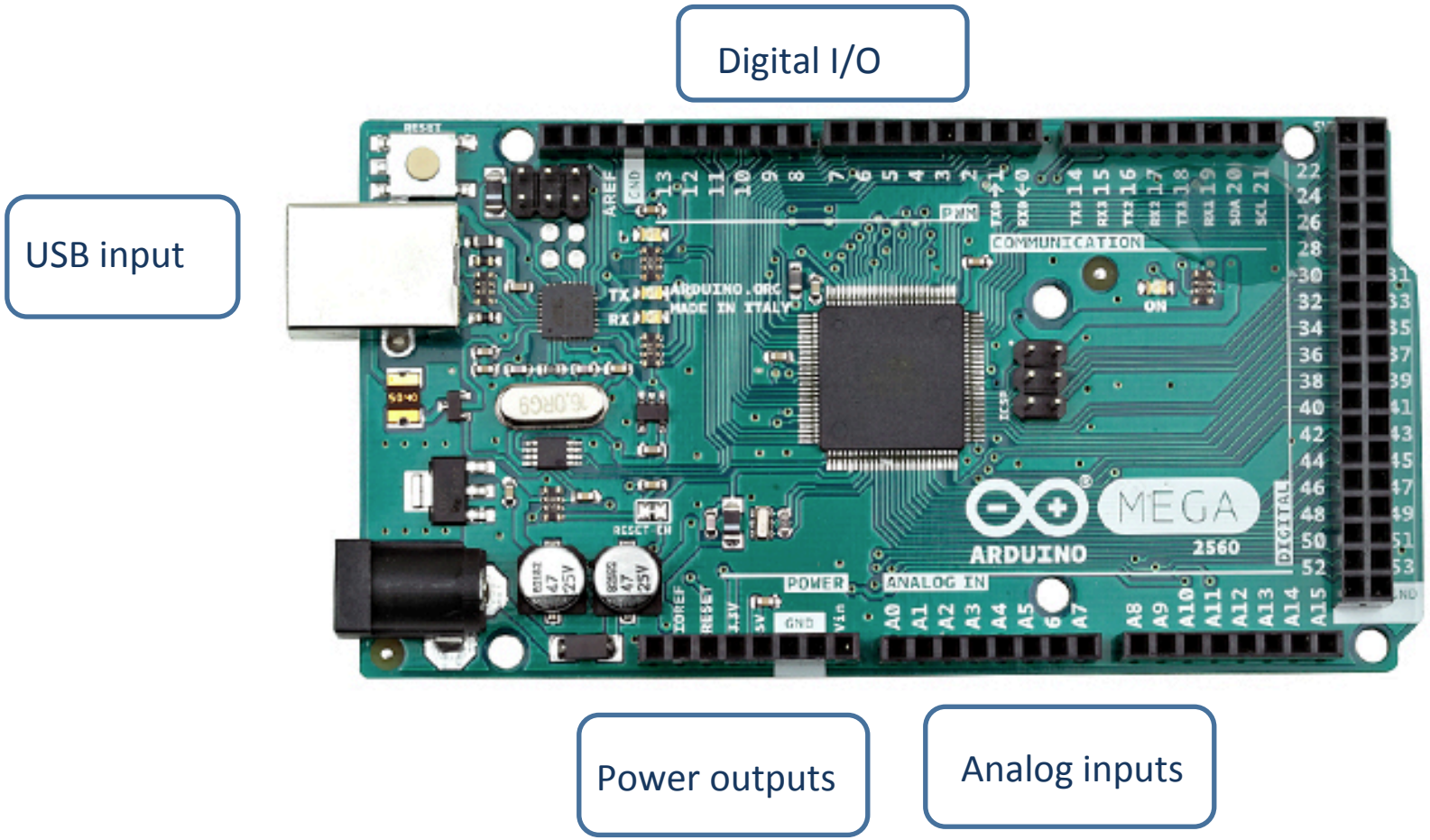


- The drive pulley attached to the the stepper motor can be seen on the right and the attachment clamp for the pendulum rod can be seen on the left.

Inverted pendulum microcontroller and stepper driver



Arduino Mega connections



Encoder attached to DI interrupt pins: encoderPinA = 18, encoderPinB = 19
Hall sensors attached to DI interrupt pins: HallSensorLeft = 20, HallSensorRight = 21
Stepper controller attached DO to pins: dirPin = 4, stepPin = 5, enPin = 6

Arduino Mega summary

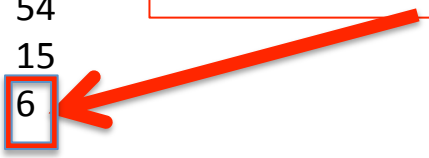
- Microcontroller — ATmega2560
- Operating Voltage — 5V
- Input Voltage (recommended) — 7-12V
- Power via the USB connection

- Digital I/O Pins — 54
- Digital I/O Pins which provide PWM output — 15
- Digital I/O Pins which provide interrupts — 6

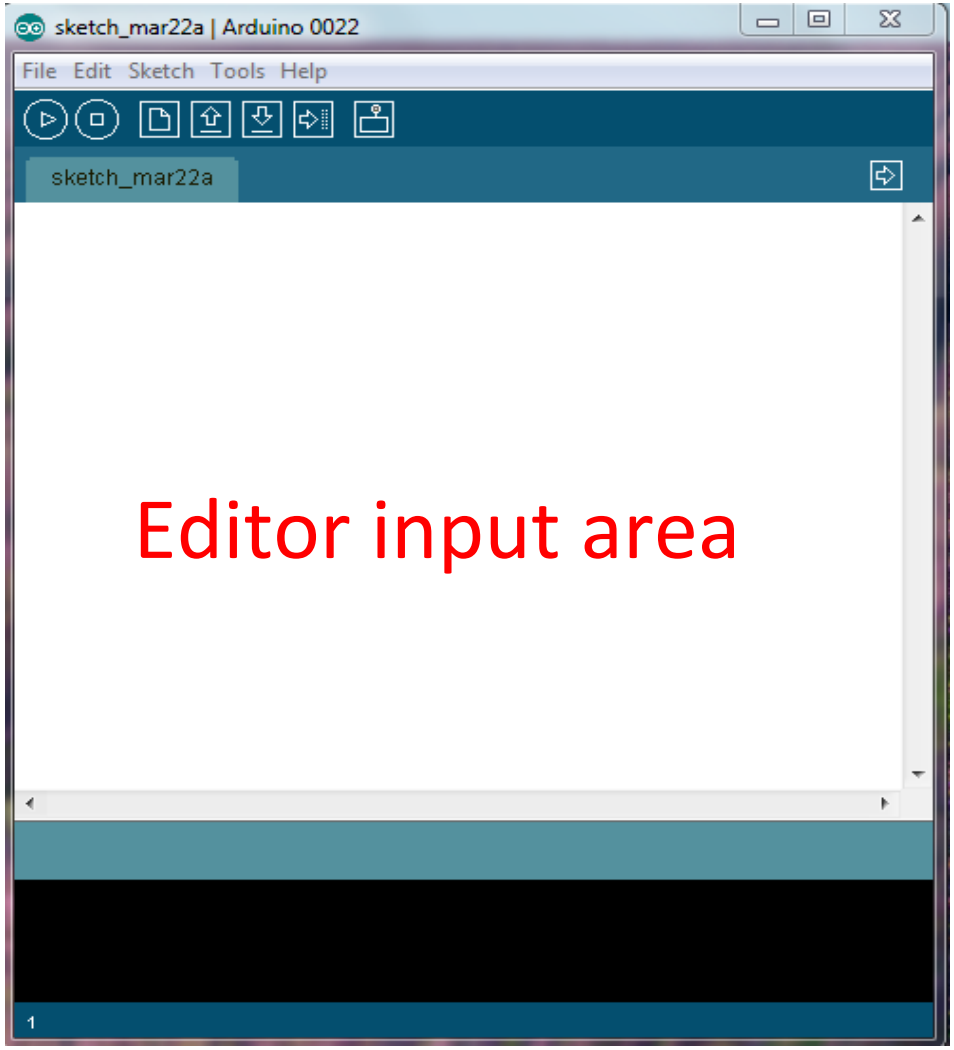
- Analog Input Pins — 16
- DC Current per I/O Pin — 40 mA
- DC Current for 3.3V Pin — 50 mA
- Operate at 5 volts
- Pins have internal pull-up resistor (disconnected by default) of 20-50 kOhms

- Flash Memory (of which 0.5 KB used by bootloader) — 256KB
- SRAM — 8 KB
- EEPROM — 4 KB
- Clock Speed — 16 MHz

Reason we need Mega and cannot use Uno



How to start coding: Software setup



- Open the Arduino main window shown on the left
- Download is available at www.arduino.cc
- Use a USB A to B cable to plug you computer into the Arduino
- Setup the port connection
- Setup the board type

Programming the Arduino

- Arduino's programming language is a version of C
- Most C commands work in the sketch editor

- Four basic components of Arduino program
  - Initialization
  - Setup
  - Loop
  - User defined functions

- Arduino Mega is designed to allow it to be reset by software running on a connected computer

Programming the Arduino: Initialization

- Include any extra libraries you are using
- All variables need to be initialized
- A=4; will not work int a= 4; will
- Define variables you will assign later
  - Int a,b,c;
- Remember data structures
  - Int, double, float, char, string ….

Programming the Arduino: setup

- Initialization for the Arduino board
- Usually begin serial communication
- Usually assign digital pins to input or output
- For example:

```
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(12, INPUT);
  Serial.begin(9600);
}
```

Void means setup will not output anything

pinMode defines the state of digital pins to OUPUT or INPUT.

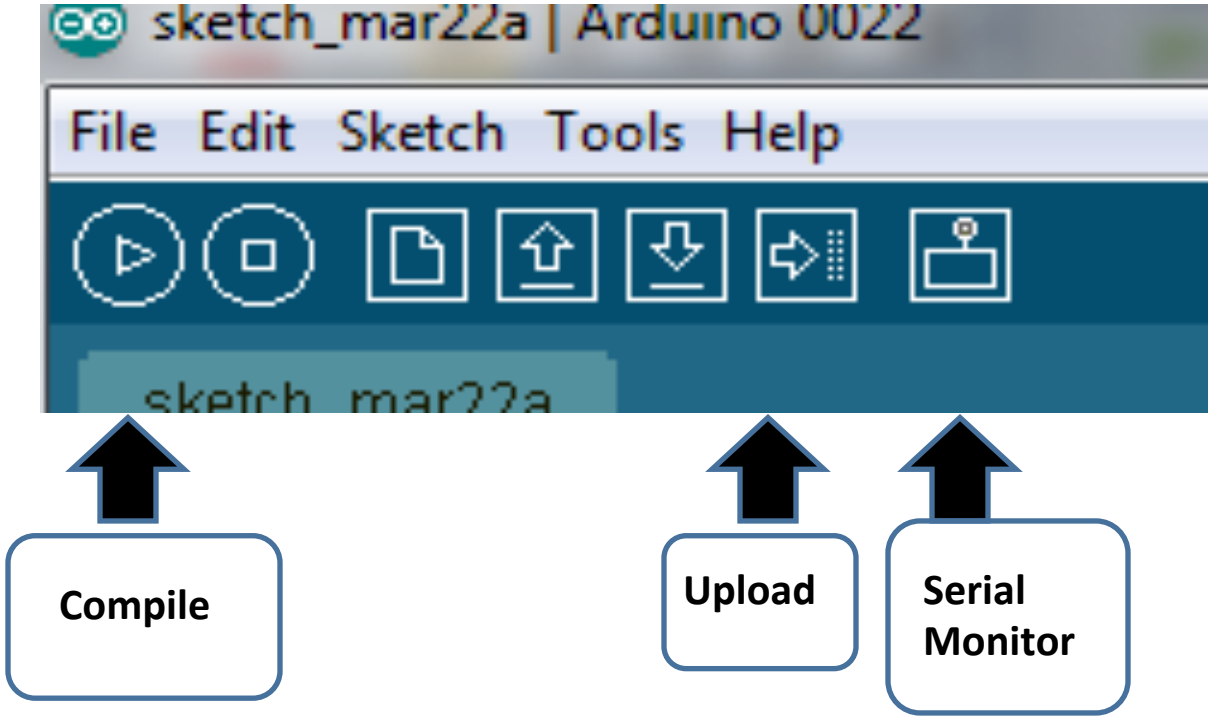
Serial.begin(9600) opens serial communication at 9600 baud

Programming the Arduino: loop

- Loop is the main portion of the Arduino code
- Loop is what the microcontroller will do forever
- This is very much like a never ending for loop

```
void loop()
{
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

Compiling and uploading code



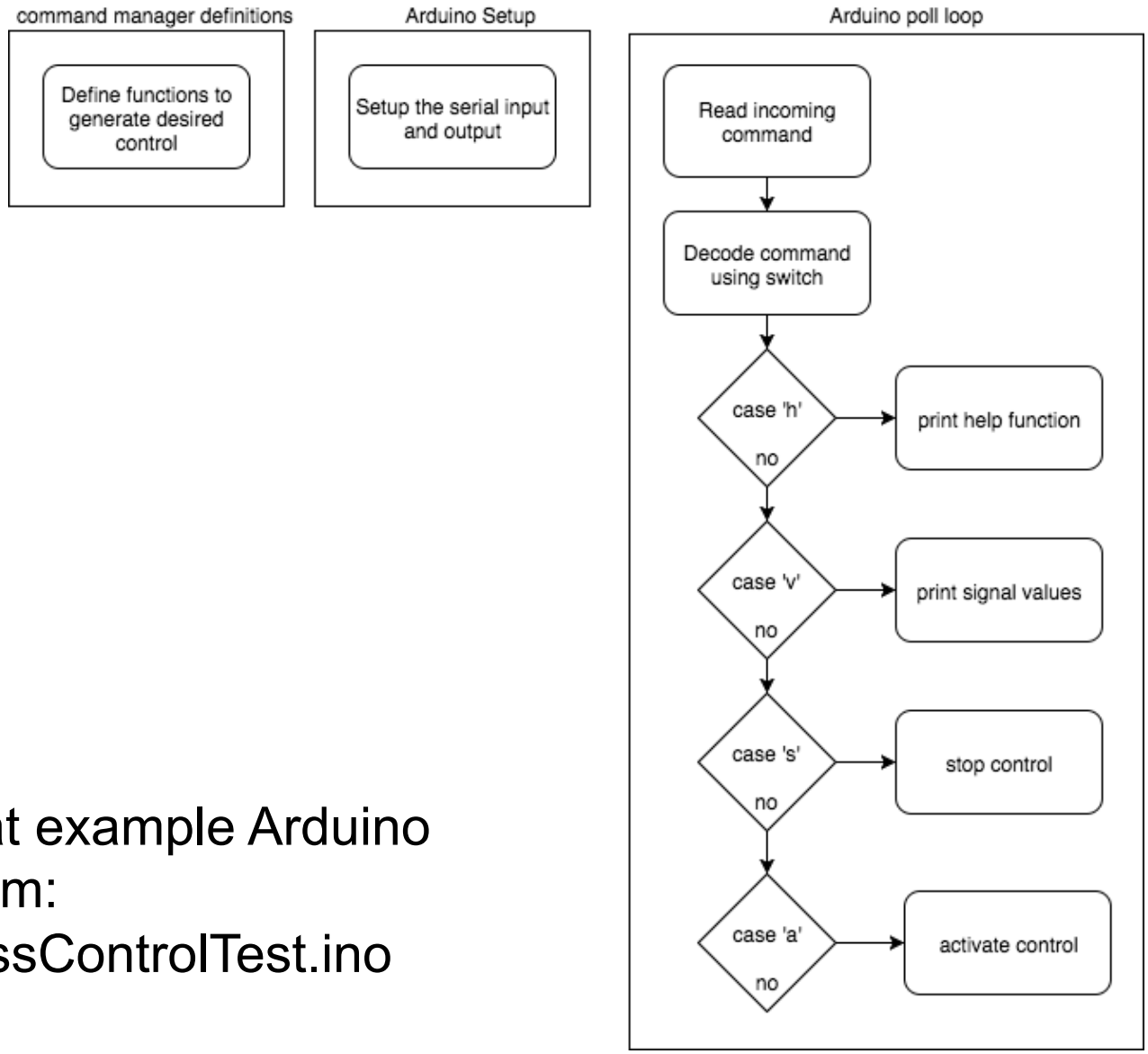
Under tools make sure Board is the board you and using and Serial port is checked and is the correct port

ROCO218: Control Engineering
Dr Ian Howard

Lecture 8

Simple Arduino example programs implementing functionality needed to control the inverted pendulum

Menu control test program flow chart



Look at example Arduino program: ProcessControlTest.ino

Menu control test program

```
// command functions activated my menu
void StopControl() {
  Serial.println("StopControl");
  // Stop Control code goes here ...
}
void ActivateControl() {
  Serial.println("ActivateControl");
  // ActivateControl Control code goes here ...
}

// print out help commands to serial output
void PrintHelp()
{
  // print help commands
  Serial.println("h: print help commands");
  Serial.println("v: print signal values");
  Serial.println("s: stop control");
  Serial.println("a: activate control");
}
void setup() {
  // setup serial monitor
  Serial.begin (9600);
}

void loop() {
  // run the options menu
  PollControlMenuCommands();
}
```

Define functions that are called to generate desired control

Define function to print out menu options on request

Arduino setup function

Setup the serial output

Arduino loop function

Process commands

Menu control test program

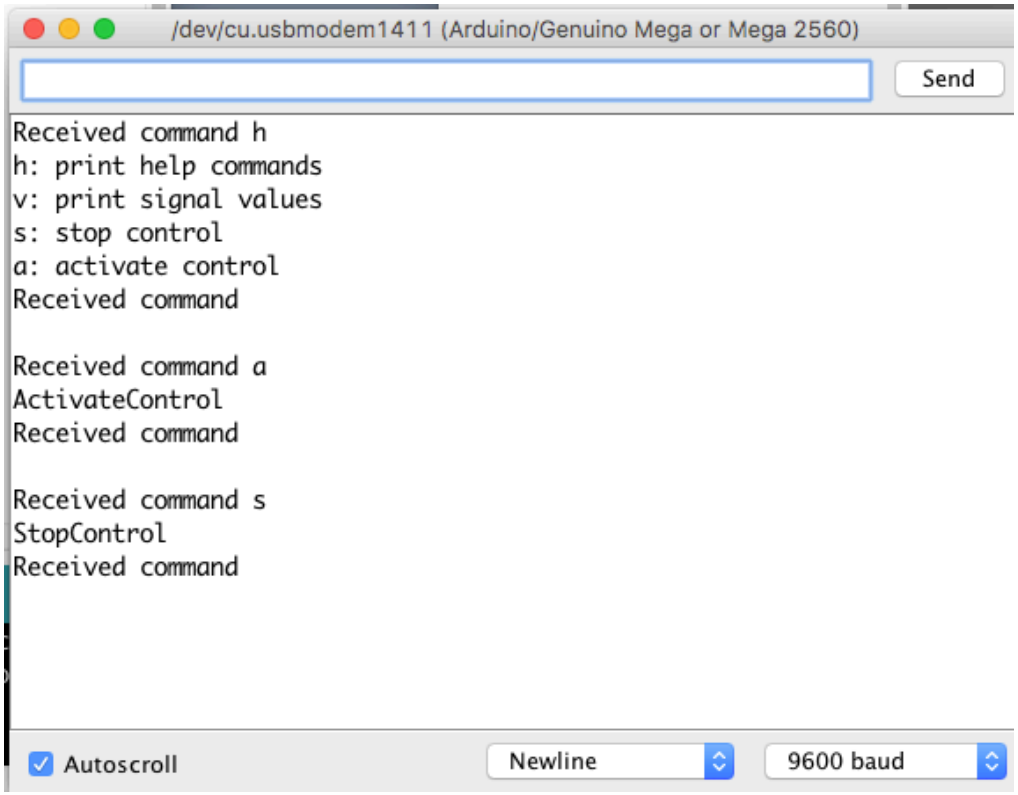
```
// decode menu input typed into the Arduino serial monitor
void PollControlMenuCommands()
{
  // check for incoming serial data:
  if (Serial.available() > 0) {

    // read incoming serial data:
    char inChar = Serial.read();
    // print out command
    Serial.print("Received command ");
    Serial.println(inChar);

    // decode the command
    switch (inChar)
    {
      // print help commands
      case 'h':
        PrintHelp();
        break;
      case 'v':
        Serial.println("print signals");
        //  output printing lines do here....
        break;
      // stop control
      case 's':
        StopControl();
        break;
      // activate control
      case 'a':
        ActivateControl();
        break;
    }
  }
}
```

If there is an incoming command

Read command and display it

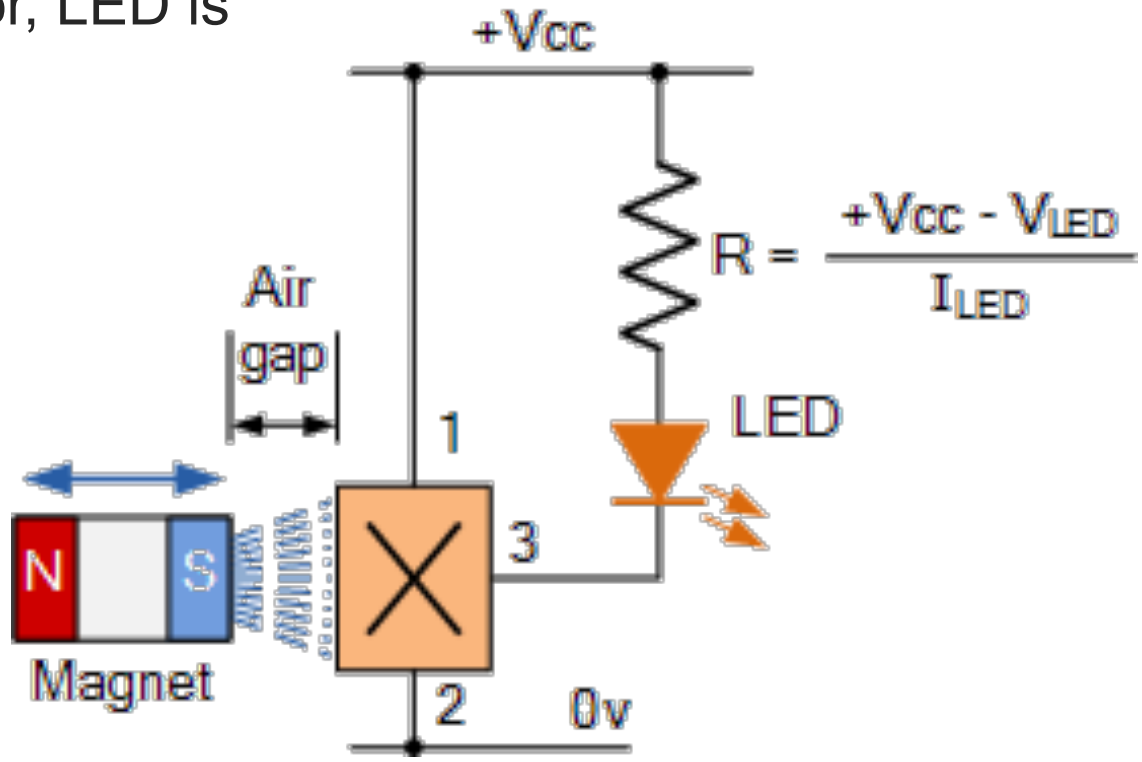
Use a switch function to decode the command and call the appropriate command function

Menu control test program

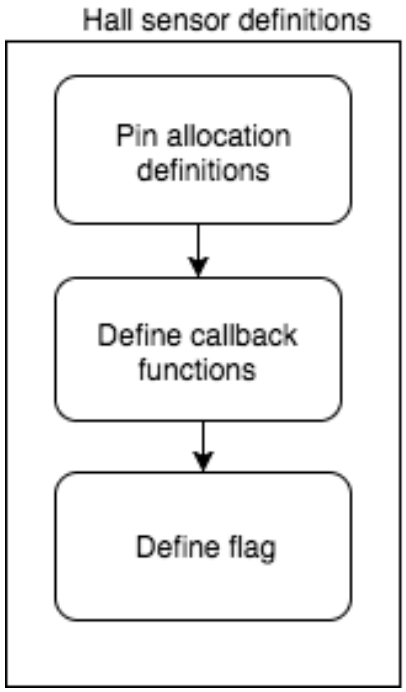


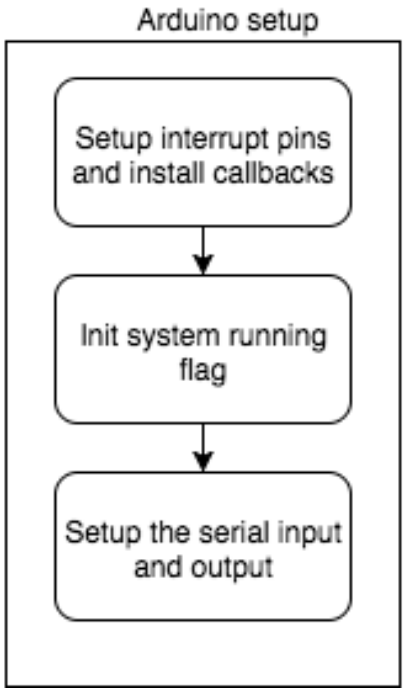
Hall effect magnetic sensor circuit

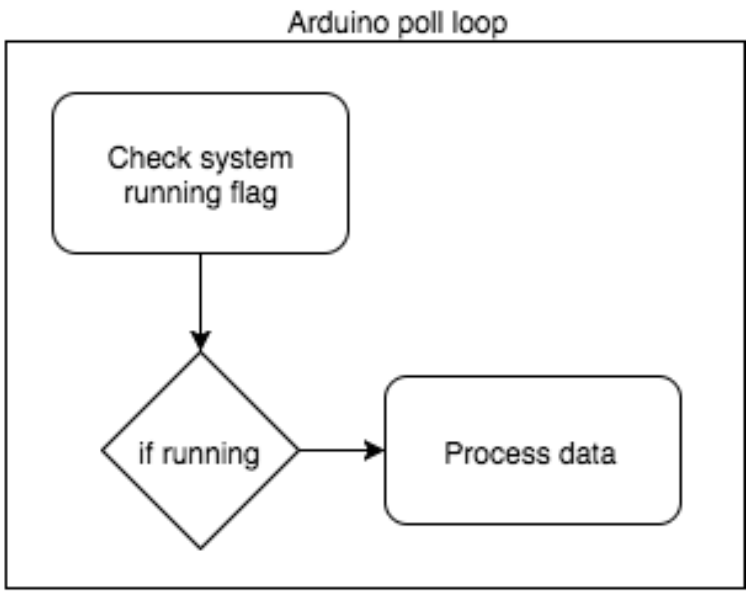
- Simple switch activated by the application of a magnetic field
- When south pole of magnet brought up to this sensor, LED is switched on



Shutdown control using Hall sensor flow chart





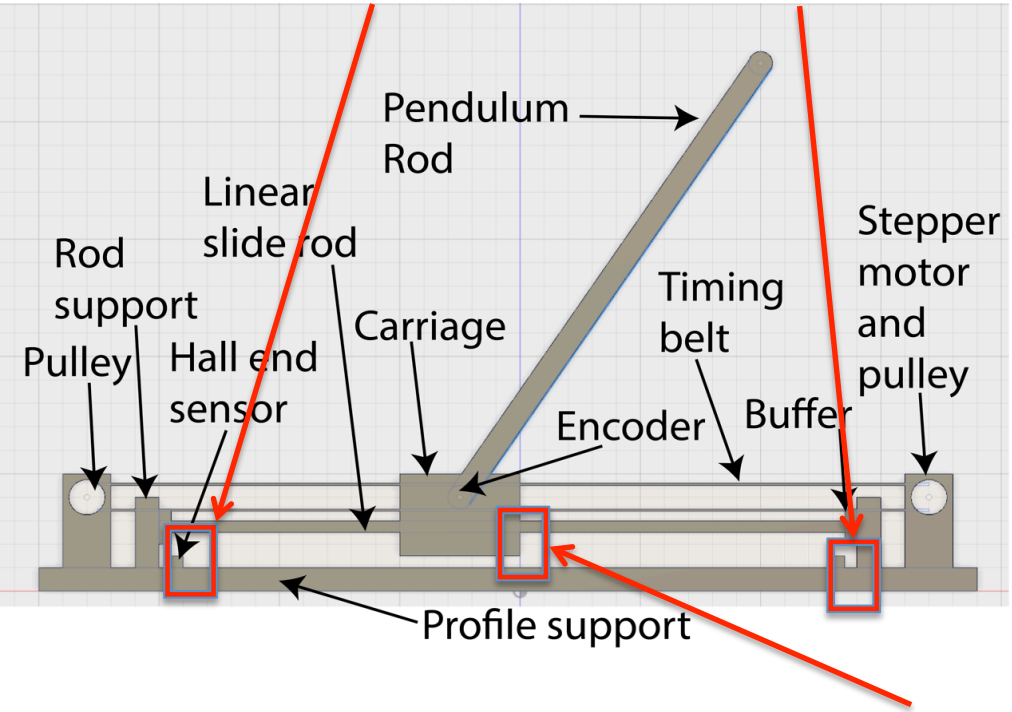


Look at example Arduino program: SimpleHallSensorTest.ino

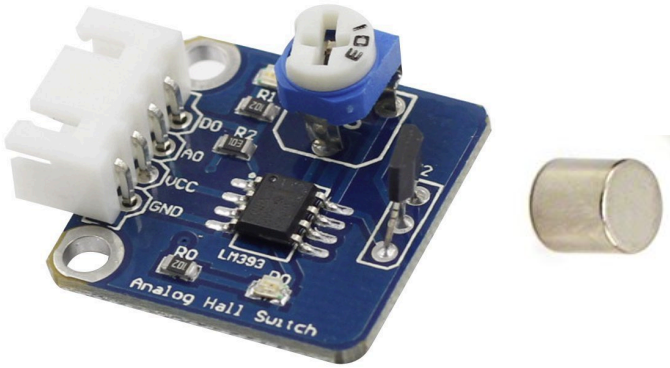
Shutdown control using Hall sensor limit switches

Sensors need to stop control when pendulum carriage reaches its end of travel to the left and to the right

Hall sensors activated by neodymium magnet on carriage



Neodymium magnet on carriage



SunFounder Analog Hall Switch Sensor Module
0v to GND
+5v power to VCC
Digital output on D0

Shutdown control using Hall sensor limit switches

```
// setup Hall endstop sensors
// control pins for Hall end stop sensors
int HallSensorLeft = 20;
int HallSensorRight = 21;

// flag to indicate if control running
bool controlRunnning;

// stop control can call running flags etc
void StopControl()
{
  // message
  Serial.println("Shut down control");
  controlRunnning = false;
}

void HallLeftCB   (void)
{
  // print message
  Serial.println("Left Hall Sensor activated");
  // call function to deactivate control
  StopControl();
}
void HallRightCB   (void)
{
  // print message
  Serial.println("Right Hall Sensor activated");
  // call function to deactivate control
  StopControl();
}
```

Define digital inputs
Use pins that support interrupts

Define flag

Define control stop function

Left sensor callback function

Right sensor callback function

Shutdown control using Hall sensor limit switches

```
// associate callback functions with interrupt pins
void SetupHall()
{
  // Setup callback input for Left Hall sensor
  pinMode(HallSensorLeft, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(HallSensorLeft), HallLeftCB, FALLING );

  // Setup callback input for Right Hall sensor
  pinMode(HallSensorRight, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(HallSensorRight), HallRightCB, FALLING );
}

void setup() {
  // setup serial monitor
  Serial.begin (9600);

  // setup sensor callbacks
   SetupHall();

  // start with control is running
  controlRunnning = true;
}

void loop() {
  // poll loop runs only if controlRunnning flag active
  if (controlRunnning == false) {
    return;
  }

  // real active polling functionality would go here
  Serial.println("Running control");

  // 1s  delay so dont print too many outputs messages
  delay(1000);
}
```

Associate Hall callbacks with pins

Arduino setup function

Use serial interface

Setup Hall sensors

Enable processing

Arduino loop function

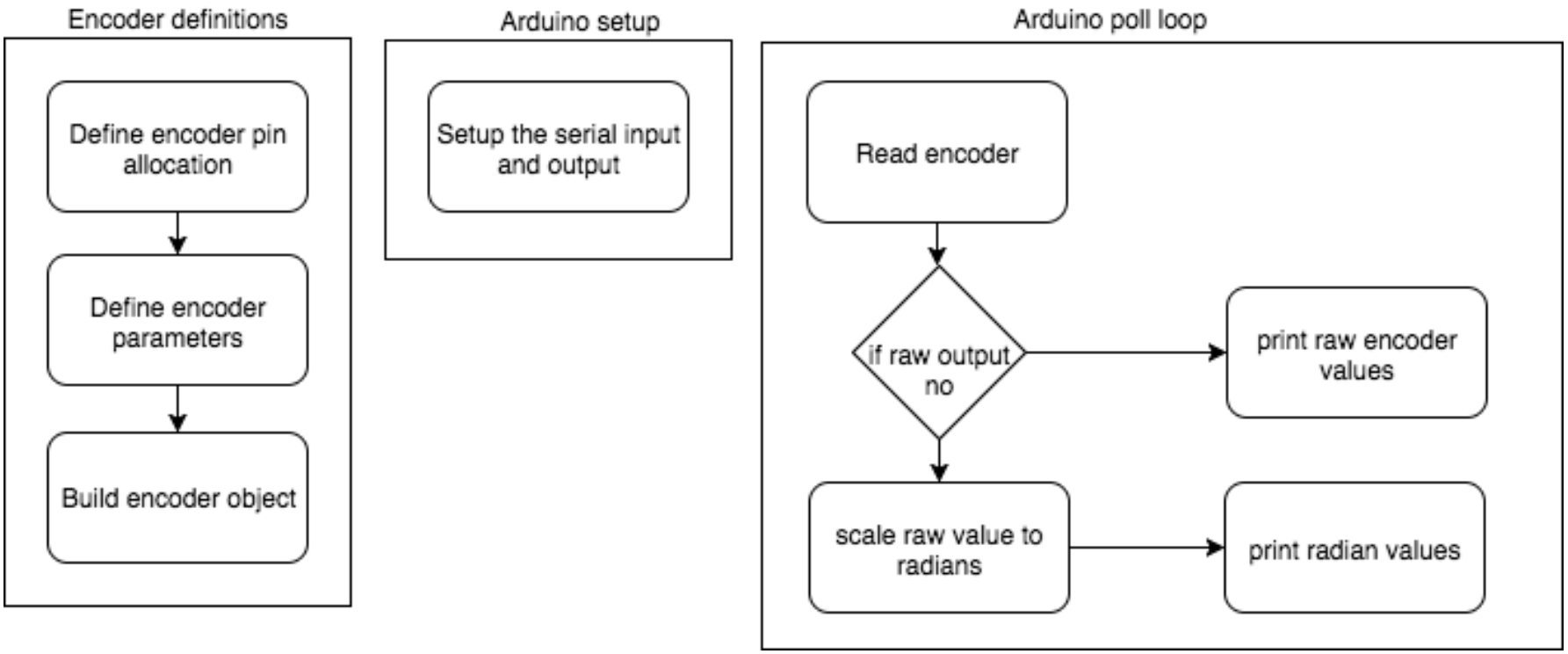
Only perform processing if flag active

Processing goes here

Digital incremental encoder

- Generates a relative position signal suitable for positioning tasks
- Rotation direction recognition
- Speed information from number of pulses per time unit
- Standard solution for many applications
- Single output types only used to determining the speed

Read encoder flow chart



Look at example Arduino program: SimpleEncoderTest.ino

Arduino incremental encoder test program

```
// setup the encoder
#include "Encoder.h"

// control pins for encoder
// Green = A phase
// white = B phase
// red = power +ve
// black = power -ve
// digital connection pints used
int encoderPinA = 18;
int encoderPinB = 19;
// set the resolution
//long encoderResolution = 4 * 600;
long encoderResolution = 4 * 2000;

// build the encoder object
Encoder myEnc(encoderPinA, encoderPinB);
```

Include the encoder class

Define digital input pins that support interrupts to connect to encoder

Build the encoder object as global variable

Arduino incremental encoder test program

```
void setup() {
  // setup serial monitor
  Serial.begin (9600);
}

// flag to select output format
bool wantRawValue = true;
void loop() {

   // 100-ms delay
  delay(100);

  // read encoder and print out values
  double encoderPosition = myEnc.read();

  if (wantRawValue) {
    // print out raw encoder values
    Serial.println(encoderPosition);
  }
  else {
    // convert raw value to radians
    double encoderAngle = encoderPosition * 2.0 * 3.142 / encoderResolution;
    // print out encoder angle in radians
    Serial.println(encoderAngle);
  }
}
```

Arduino setup function

Use serial interface

Arduino loop function

100ms delay so doesn't run too fast

Read encode

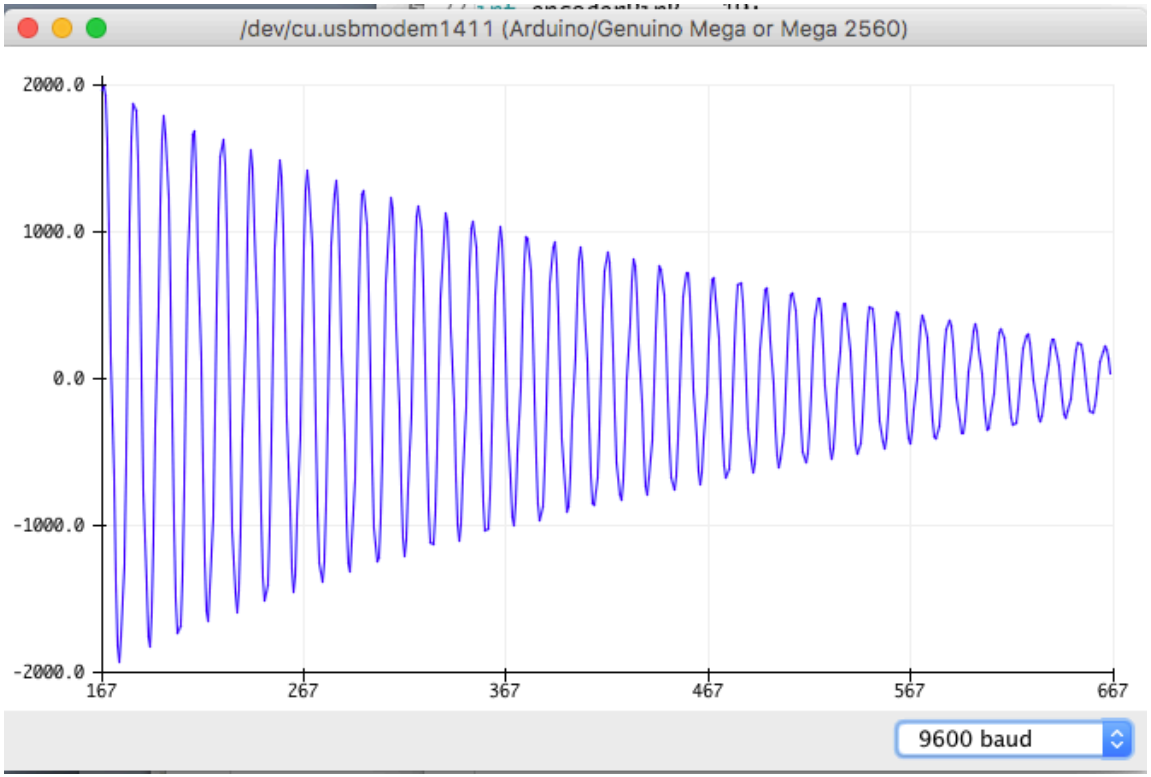
Choose output format

Print out raw encoder count

Print out angle in radians

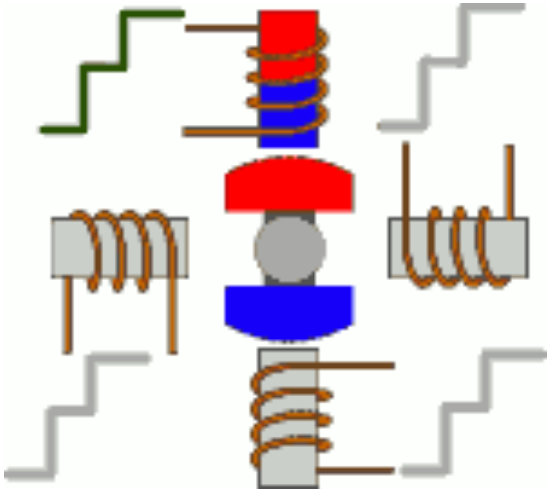
Arduino incremental encoder test program

Plot output using serial monitor

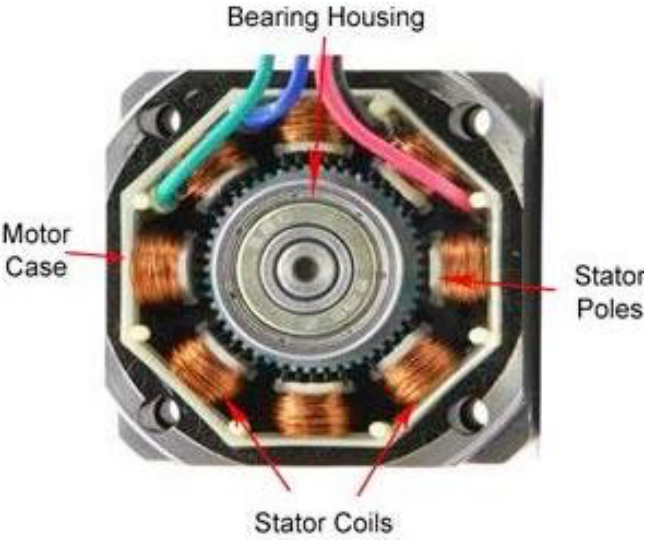


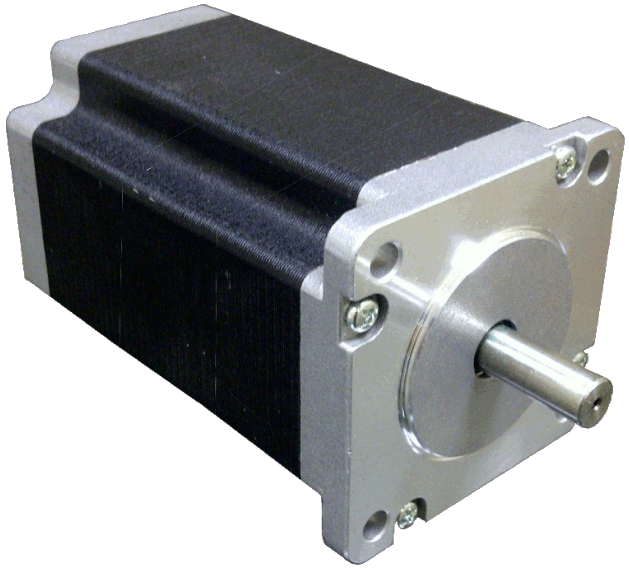
Microstep stepper motor control

- Use micro step mode
- This implements a gradual transition between pole positions
- This leads to smoother operation that full-step

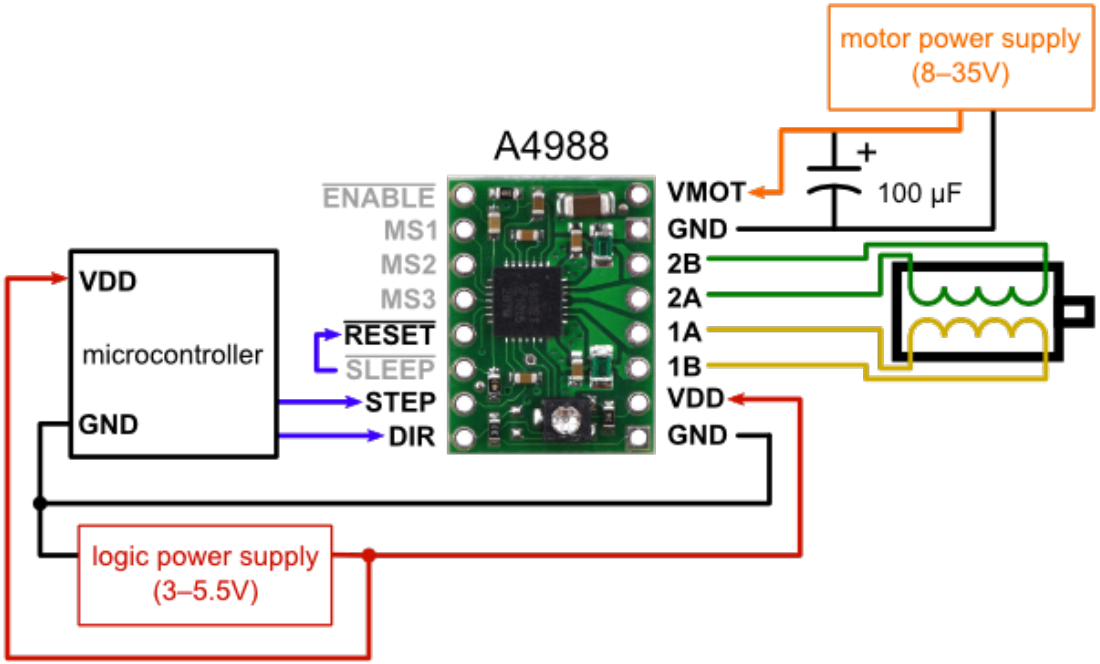


Microstepping

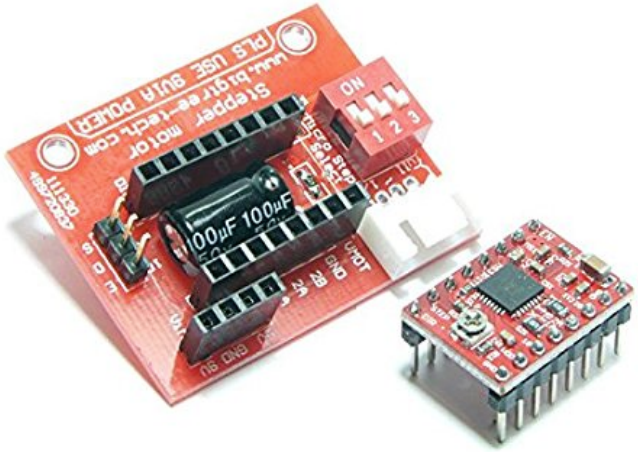




A4988 Stepper Motor Controller

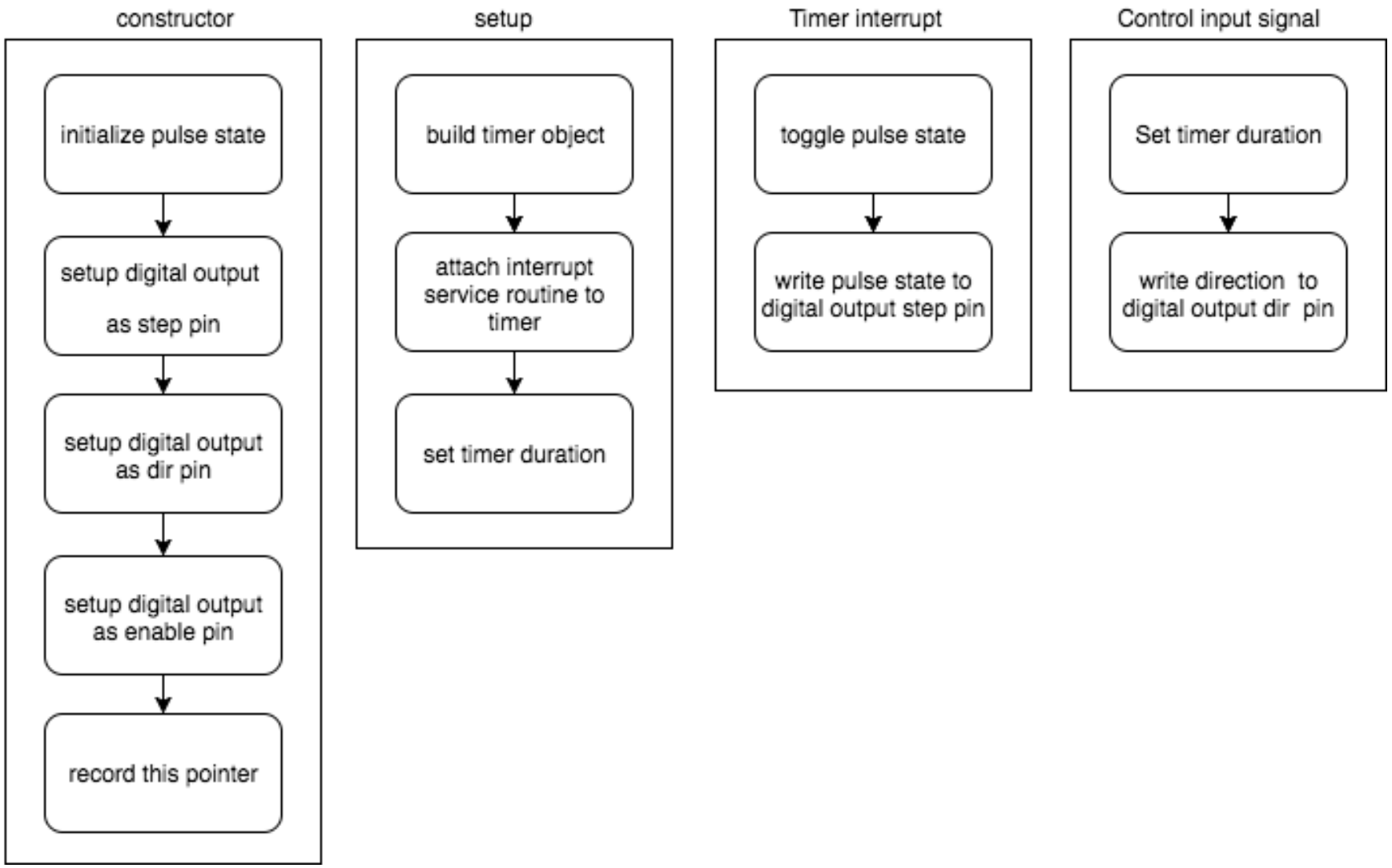


Stepper shield



- Intelligent current control up to 2A
- Set stepping mode via control pins – set to 4x micro stepping
- Simple step and direction control interface
- To operate: Need to generate
  - control pulses (1 pulse per micro step)
  - direction signals (so motor turns in desired direction)
- https://www.pololu.com/product/1182

CStepper velocity control class flow chart



CStepper velocity control class flow chart

```
// construction
CStepper::CStepper(int stepPin, int dirPin, int enPin) {

  // set parameters
  this->stepPin = stepPin;
  this->dirPin = dirPin;
  this->enPin = enPin;

  // init
  pulseState = LOW;
  pulseCount = 0;
  cartLeftDirection = 1;

  // setup stepPin pin for each stepper
  pinMode(stepPin, OUTPUT);

  // setup dirPin pin for each stepper
  pinMode(dirPin, OUTPUT);

  // setup enable pin for each stepper
  pinMode(enPin, OUTPUT);

  // record this for callback acccess to object
  this_g = this;
}

// destruction
CStepper::~CStepper() {
}
```

Define pins used for stepper

Define signals variables

Setup digital output pins

Record this pointer
needed for callback

CStepper velocity control class flow chart

```
// pulse genrator ISR
// 50% duty cycle achieve by toggling output each time called

// global this pointer for object access in callbacks
static CStepper * this_g;

// static stepper interrupt service callback routine
static void pulseISR()
{
  // run replay
  this_g->RunPulse();
}
```

Define this pointer

Define interrupt service routine that will generate pulses

```
// setup the timer
void CStepper::SetupTimer()
{
  // This Timer1 function must be called first
  // Argument in "microseconds" is the period of time the timer takes
  // run every 1 second
  Timer1.initialize(150000);
  Timer1.attachInterrupt(pulseISR);

  // set cartLeftDirection = 1
  SetStepperParams(150000, 1);
}
```

Setup a timer

Attach interrupt routine that will be called by timer

CStepper velocity control class flow chart

```
// pulse processing function called from ISR
void CStepper::RunPulse() {

  // toggle pulse
  if (pulseState == LOW) {
    pulseState = HIGH;
    // keep track of pulses
    if (cartLeftDirection == 1)
    {
      // count pulses
      pulseCount--;
    }
    else
    {
      // count pulses
      pulseCount++;
    }
  }
  else {
    pulseState = LOW;
  }
  digitalWrite(stepPin, pulseState);
}
```

Pulse generation function

Toggle pulse state

Inc or dec pulse count depending on direction

Write pulse state to digital output

CStepper velocity control class flow chart

```
// switch on steppers
void CStepper::EnableSteppers()
{
  // reset values
  pulseState = LOW;
  pulseCount = 0;

  // enable the outout
  digitalWrite(enPin, LOW);
}

// switch off steppers
void CStepper::DisableSteppers()
{
  // disable the outputt
  digitalWrite(enPin, HIGH);
}
```

Enable the stepper

Disable the stepper

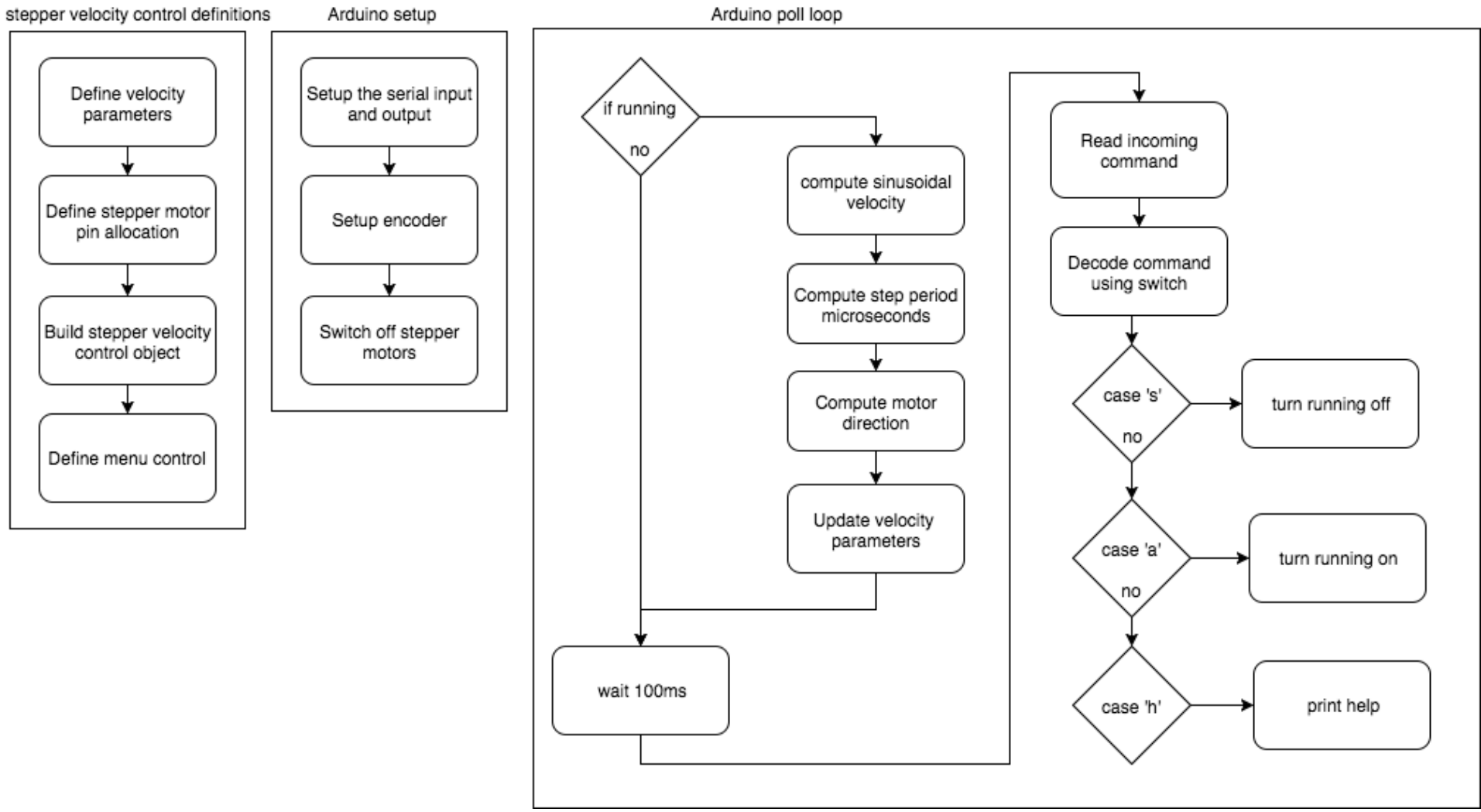
```
// Set the stepper motor pulse period
void CStepper::SetStepperParams(long microseconds, int cartLeftDirection)
{
  // record direction
  this->cartLeftDirection = cartLeftDirection;

  // generate output
  if (cartLeftDirection == 1) {
    digitalWrite(dirPin, HIGH);
  }
  else
  {
    digitalWrite(dirPin, LOW);
  }

  // Set a new period
  Timer1.setPeriod(microseconds);
}
```

Stepper Function

Set stepper direction

set timer period

Arduino stepper velocity control flow chart



Arduino stepper velocity control test program

```
// Velocity control of stepper
#include "CStepper.h"
#include "math.h"

// set movement to 0.5Hz
double carriageFrequency = 0.5;

// set peak speed to 10cm/s
double carriageAmplitide = 0.1;

// running time
double timeIdx;

////////////////////////////////////////////////////////////////
// stepper motor parameters

// control pins for A4988
int stepPin = 5;
int dirPin = 4;
int enPin = 6;

// stepper accouting for 4x microstepping
double stepperPPR = 4 * 200;
double pulleyCircumference = 0.113;

// setup stepper motor control obkject
CStepper myStepper(stepPin, dirPin, enPin);
```

Include the stepper class

Define the movement parameters

Define the encoder pins

Define stepper motor parameters

Build the velocity control stepper motor object

Arduino stepper velocity control test program

```
void setup() {
  // setup serial monitor
  Serial.begin (9600);

  // setup stepper timer routine
  myStepper.SetupTimer();

  // turn the steppers off
  myStepper.DisableSteppers();
  Serial.println("Steppers disabled");
}

void loop() {

  // run the stepper speed controller loop
  PollStepperControl(0.01);

  // wait for a 10-ms delay
  delay(10);

  // run the options menu
  PollControlMenuCommands();
}
```

Arduino setup function

Setup serial I/O

Setup timer

Turn off the steppers to start with

Arduino loop function

Regularly poll the stepper control

Put in 10ms delay

Regularly poll the command manager

Arduino stepper velocity control test program

```
// poll loop function to decode menu input typed into the Arduino serial monitor
void PollControlMenuCommands()
{
  // check for incoming serial data:
  if (Serial.available() > 0) {

    // read incoming serial data:
    char inChar = Serial.read();

    // print out command
    Serial.print("Received command ");
    Serial.println(inChar);

    // decode the command
    switch (inChar)
    {
      // print help commands
      case 'h':
        PrintHelp();
        break;
      // switch off steppers and control
      case 'o':
        myStepper.DisableSteppers();
        Serial.println("Steppers deactivated");
        break;
      // activate or reactivate steppers and control
      case 'a':
        myStepper.EnableSteppers();
        Serial.println("Steppers activated");
        timeIdx=0;
        break;
    }
  }
}
```

Poll function for command manager

Read and echo back incoming command

Decode the commands

Report help option

Switch off stepper option

Switch on stepper option

Arduino stepper velocity control test program

```
// run the stepper speed controller loop
void PollStepperControl(double timeInc)
{
  double cartLeftDirection;

  // increment time
  timeIdx = timeIdx + timeInc;

  // calculate instantaneous velocity
  double omegaT =  2 * 2.0 * 3.142 * carriageFrequency * timeIdx;
  double shortTermVel =  carriageAmplitide * sin(omegaT);

  // determine the motor direction
  if (shortTermVel > 0) {
    cartLeftDirection = 1;
  }
  else
  {
    cartLeftDirection = 0;
  }

  // scale output velocity signal from m/s to pulses/s
  double velocity = 2 * stepperPPR * shortTermVel / pulleyCircumference;
  //Serial.println(velocity);

  // calculate the pulse period
  double microseconds = 1000000;
  if (abs(velocity) > 0) {
    microseconds = 1000000 / abs(velocity);
  }

  // Set stepper motor period and direction
  myStepper.SetStepperParams(microseconds, cartLeftDirection);
}
```

Poll function for stepper speed generation

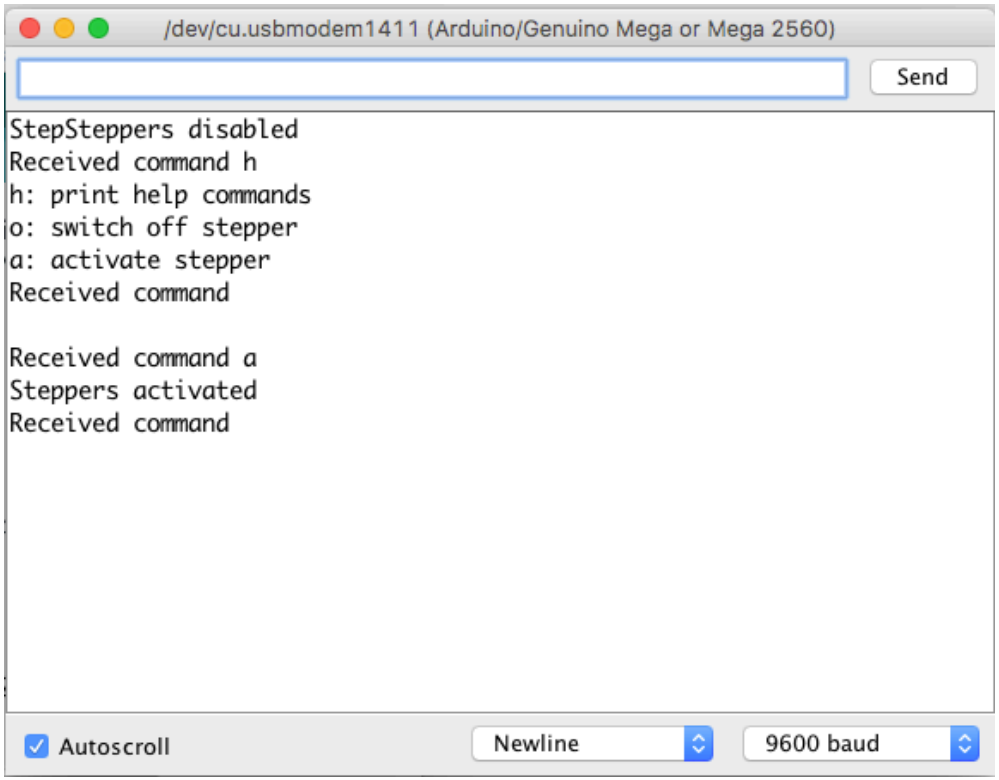
Calculate current sine wave modulated output velocity

Determine motor direction

Determine stepper pulse period needed

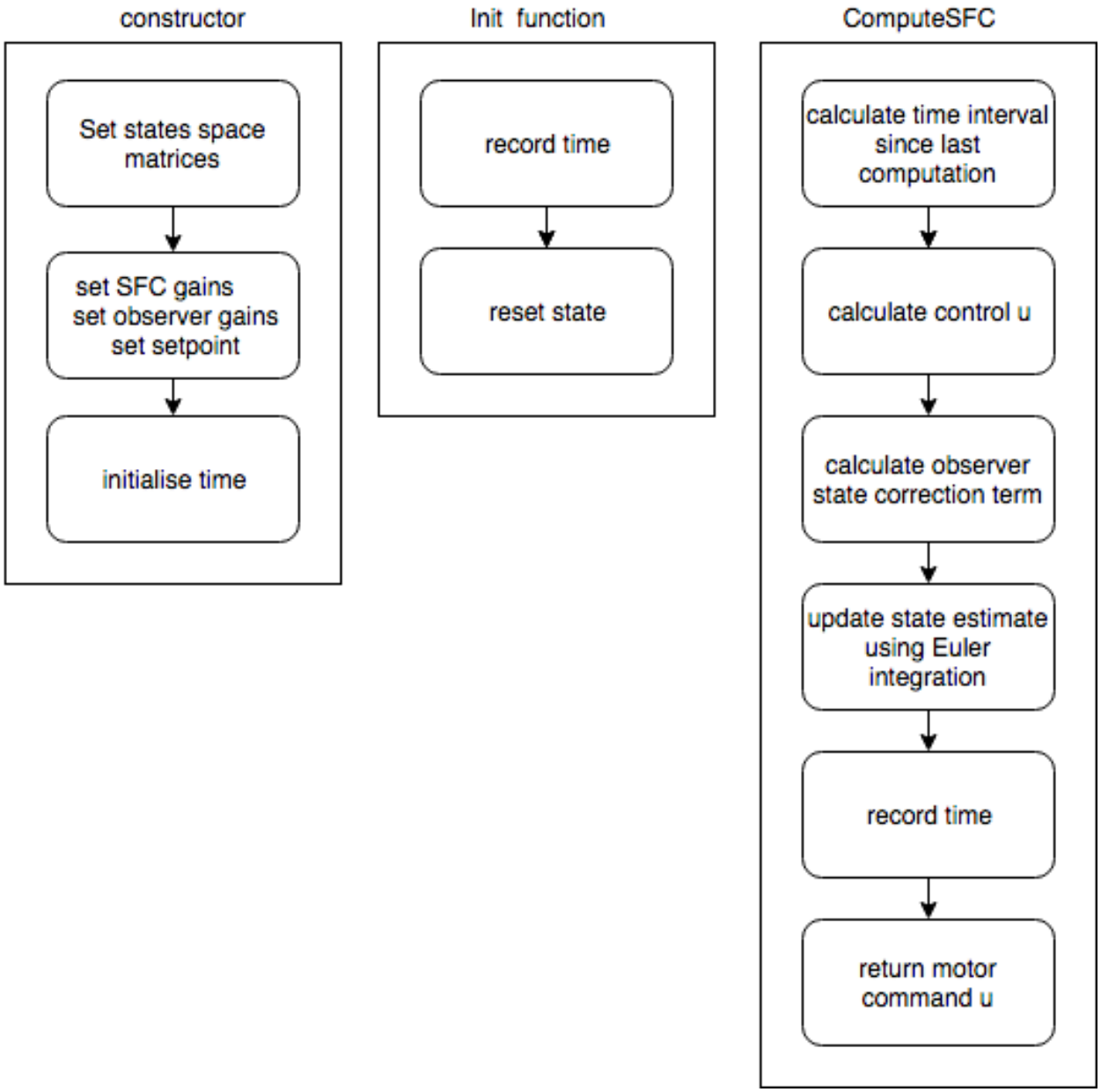
Use the pulse period and direction

Arduino stepper velocity control test program



Look at example Arduino program: StepperVelocityTest.ino

CSFC3 class control flow chart



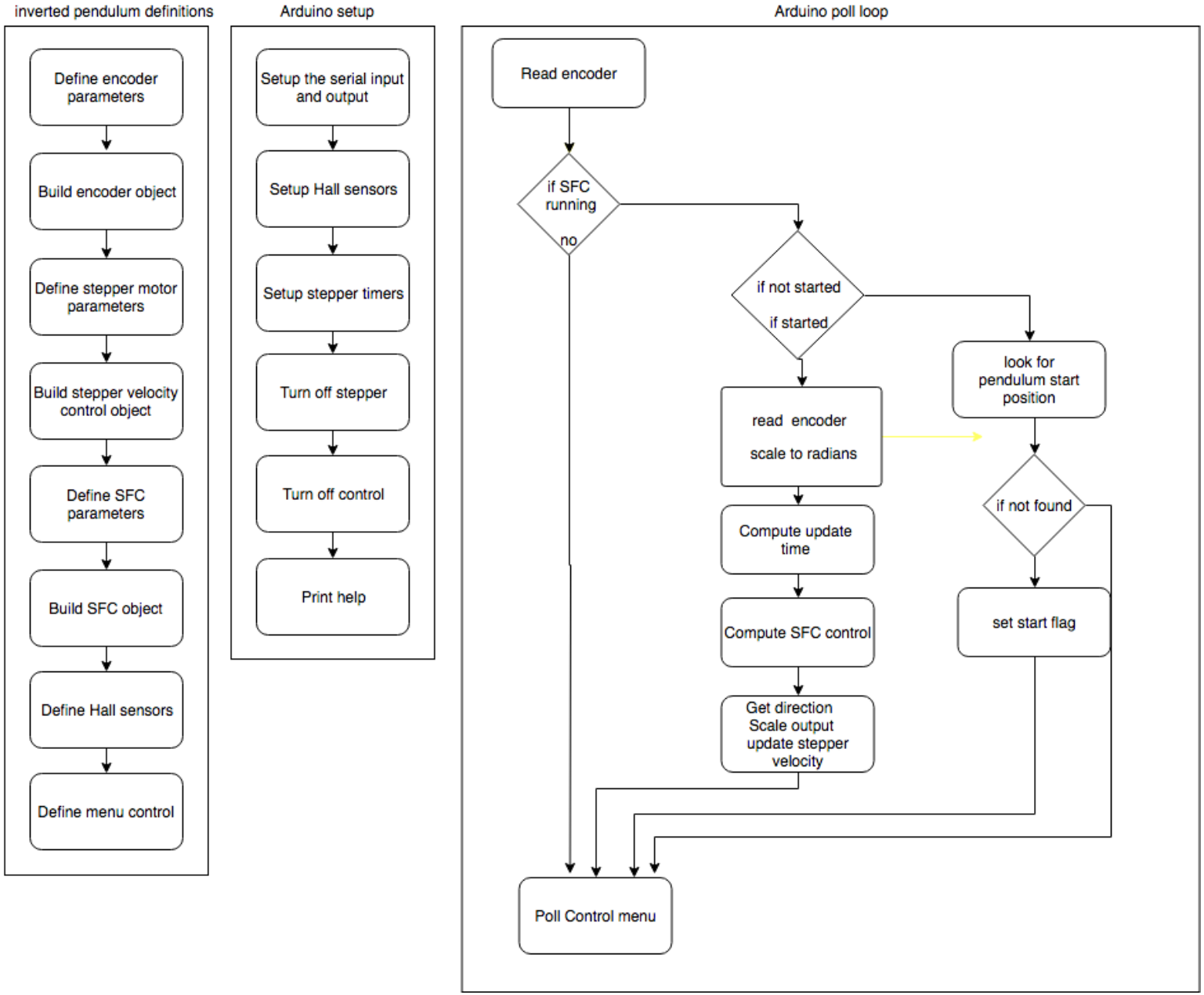
SFC with observer C++ implementation

```
// construction
CSFC3::CSFC3(double A[][3], double B[],  double C[], double K[], double L[], double setPointAngle) {

  // set rank for SFC
  rank = 3;

  // rank 2 observer!
  this->L[0] = L[0];
  this->L[1] = L[1];

  // for all rows
  for (int ridx = 0; ridx < rank; ridx++)
  {
    this->B[ridx] = B[ridx];
    this->C[ridx] = C[ridx];
    this->K[ridx] = K[ridx];

    // for all columns
    for (int cidx = 0; cidx < rank; cidx++)
    {
      this->A[ridx][cidx] = A[ridx][cidx];
    }
  }

  this->setPointAngle = setPointAngle;

  // init values
  lastTime = 0;
}

// destruction
CSFC3::~CSFC3() {

}

// Setup the SFC
void CSFC3::InitSFC(double inputAngle, unsigned long theTime)
{
  // record measure time
  this->lastTime = theTime;

  // init state
  // for all rows
  for (int ridx = 0; ridx < rank; ridx++)
  {
    xhat[ridx] = 0;
  }
}
```

SFC with observer C++ implementation

```
// compute the SFC
// given output angle of pendulum y and the current time
// returns the motor command u
double CSFC3::ComputeSFC(double y, unsigned long theTime)
{
  // control value - stepper motor speed
  double   u = 0.0;
  double ymcn;

  // Calculate time since last update
  // delta time in given in seconds
  double h = (double)(theTime - lastTime) / 1000.0;

  // compute control variable u
  u = -xhat[0] * K[0] -  xhat[1] * K[1] -  xhat[2] * K[2] ;

  // calculate observer correction term
  ymcn = (y - setPointAngle) - C[0] * xhat[0] - C[1] * xhat[1] ;

  // update the state estimates for theta and thetaHat according to the equation
  // xhat = xhat + h * (A * xhat + B * u + L * (y - C * xhat) );
  // calculate state derivative xhatdot and sum up
  xhat[0] +=  h * (A[0][0] * xhat[0] + A[0][1] * xhat[1] + B[0] * u + L[0] * ymcn );
  xhat[1] +=  h * (A[1][0] * xhat[0] + A[1][1] * xhat[1] + B[1] * u + L[1] * ymcn );

  //  use control velocity from input to update position of cart
  xhat[2] +=  h * (B[2] * u  );

  // record variables
  lastTime = theTime;

  // return motor command
  return (u);
}
```

SFC inverted pendulum control flow chart



Arduino inverted pendulum state space control

```
void loop()
{
  // read the raw encoder value
  double encoderPosition = myEnc.read();

  // only run the SFC controller loop if active
  if (pendulumActive == true) {

    // Starts when desired equilibrium position reached
    if (started == false)
    {
      // if not started look for start condition
      PollStart(encoderPosition);
    }
    // Else if balancing started
    else
    {
      // otherwise run SFC poll loop
      // scale input to radians
      double inputAngle = ScaleInput(encoderPosition);

      // measure the time
      long int theTime = millis();

      // compute SFC output on the basis of pendulum angle
      double SFCOutput = mySFC.ComputeSFC(inputAngle, theTime);

      // set correction in appropriate direction
      SFCOutput = SFCOutput * correctionDirection;

      // Scale output to pulse duration in microseconds to achieve required velocity
      // also limit SFC output
      long microseconds = ScaleOutput(SFCOutput);

      // get mnotor direction
      int cartDirection = GetMotorDirection(SFCOutput);

      // Set stepper motor period and direction
      myStepper.SetStepperParams(microseconds, cartDirection);
    }
  }

  // Regularly service command menu
  PollControlMenuCommands();
}
```

Arduino loop function

Read encoder

If system active proceed

If balancing not started look for start

Else if balancing started

Scale encoder input and get time

Compute SFC controller output

Compute pulse period and motor direction

Set desired motor speed and direction

Regularly service command menu