# Pavo robot-side Code Information

Content of code OUTDATED:

1. Changeable and tuneable parameters
   * General actiaon execution speed
   * Empty footstep modifier
   * Start-up options (toggles initialisation or testing mode)

* Calibration for centres of servos
* Movement ranges, maximum degrees of movement for each joint

1. Movement sets
2. Not usually changed parameters and various other variable definitions
   * Servo Numbers
3. Predefined Functions and calculations
   * Servodriver Pulse calculations
   * executePosition function, Write position dataset to servos
4. assumePosition()
   * Misc setup
5. Setup
   * Serial begin at 9600 baudrate
   * Servodriver setup
6. Void Loop
   * check controller command data
   * .
   * Manual Pitch Control

## Walking set info

Final set to be written to servos is a combination of an empty set of sets, and the front back movements and left right movements are appended on to a degree determined by the fuzzy logic crisp output.

0.8 secs looks to be a good duration for a full walk cycle as of now

How do I make between different sets have different amount of divisions?

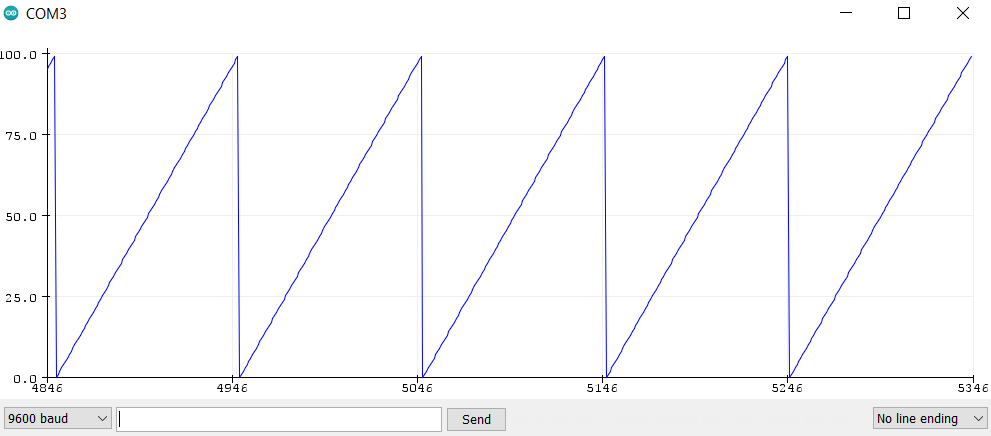
Divisions will occur at even intervals, so if I want a faster movement, I will divide a movement into fewer divisions that end up more spaced out

Sets in sets

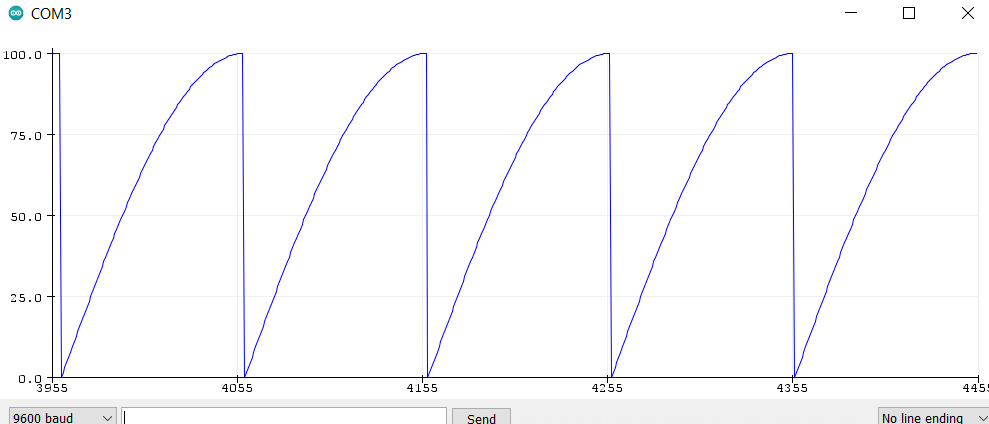
If modifiers for x and y are below a certain preset threshold, pavo will resume neutral standing positin unmovingi.e. Pavo will automatically come to rest if it is balanced

Easing function Multipliers:

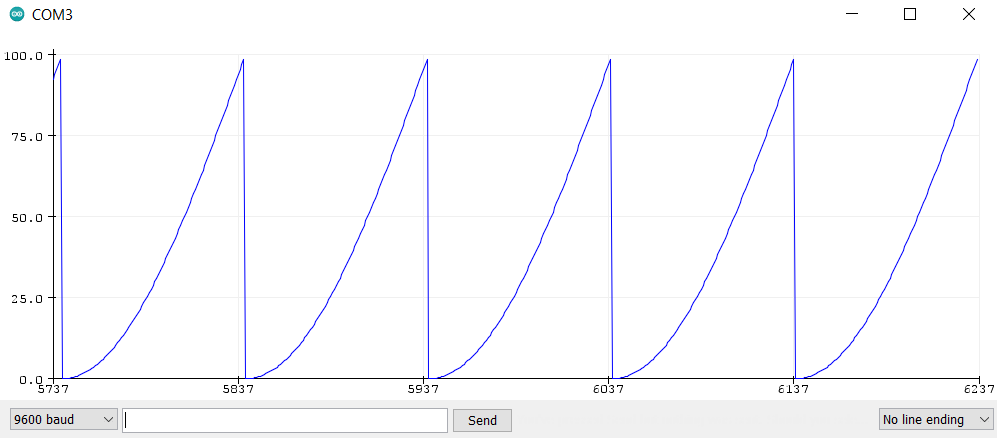
Linear:



slow down:

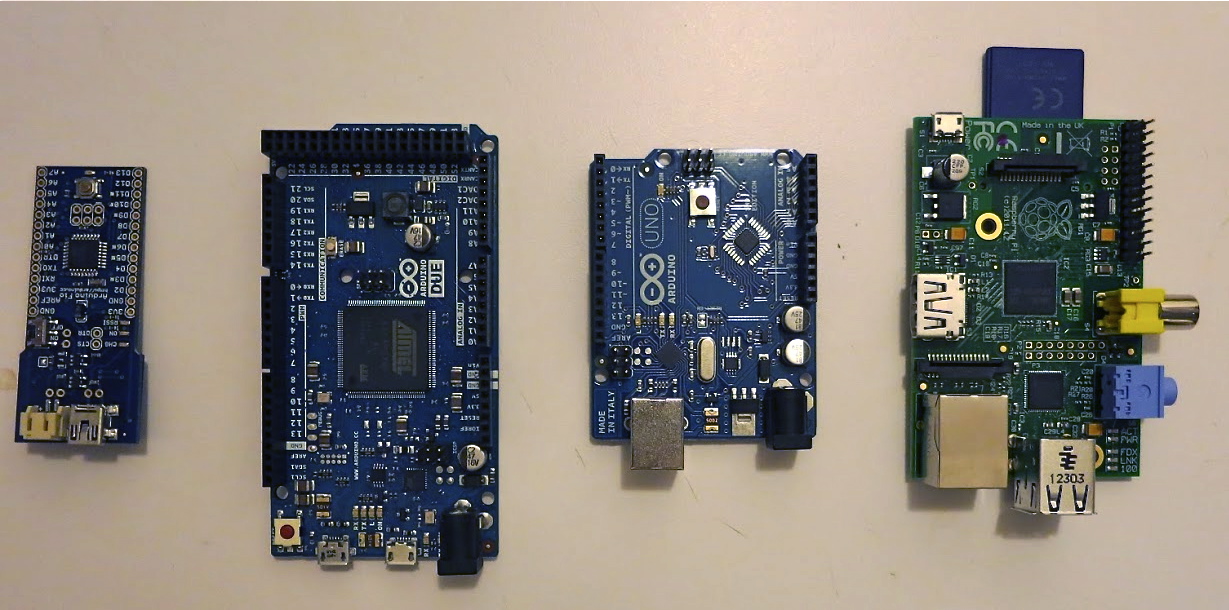


speed up:



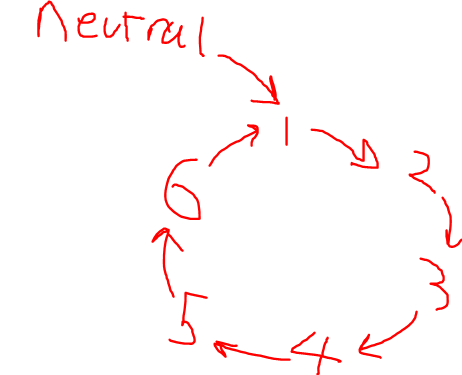
MEMORY MAXIMUM HAS BEEN REACHED

Due to the large size of sets that need to be processed and stored for footstep planning, the maximum divisions has been set to 18



The Due is not too much bigger

<https://learn.adafruit.com/memories-of-an-arduino/optimizing-sram>

Cycle Loop:

Old walking sets

//Temporary walking sets L L L R R R M M M t ---------------------------------------------------------------------

const int temp1ServoState[] = {-30, -10, 0, 80, 100, 0, 0, 0, 0, 15};

const int temp2ServoState[] = {-15, 30, 0, 10, 80, 0, 0, 0, 0, 15};

const int temp3ServoState[] = {-80, 40, 0, 10, 40, 0, 0, 0, 0, 15};

const int temp4ServoState[] = {-80, -100, 0, 30, 10, 0, 0, 0, 0, 15};

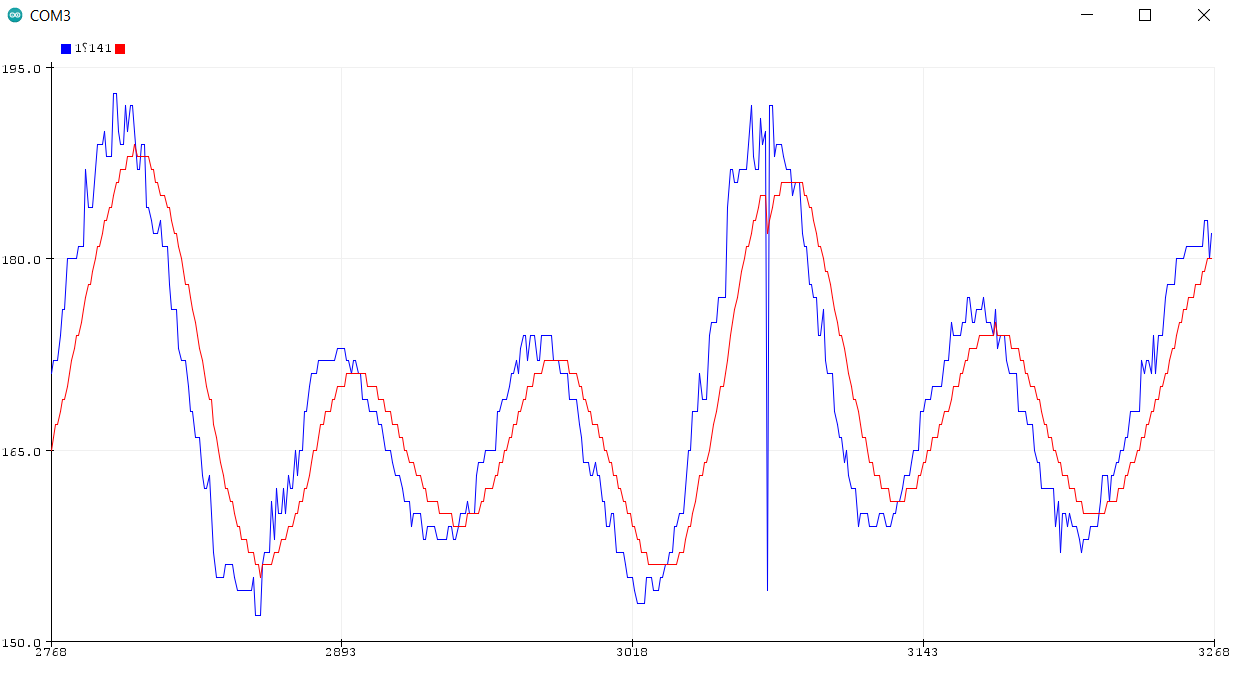
const int temp5ServoState[] = {-10, -80, 0, 15, -30, 0, 0, 0, 0, 15};

const int temp6ServoState[] = {-10, -40, 0, 80, -40, 0, 0, 0, 0, 15};

Exponential filter:

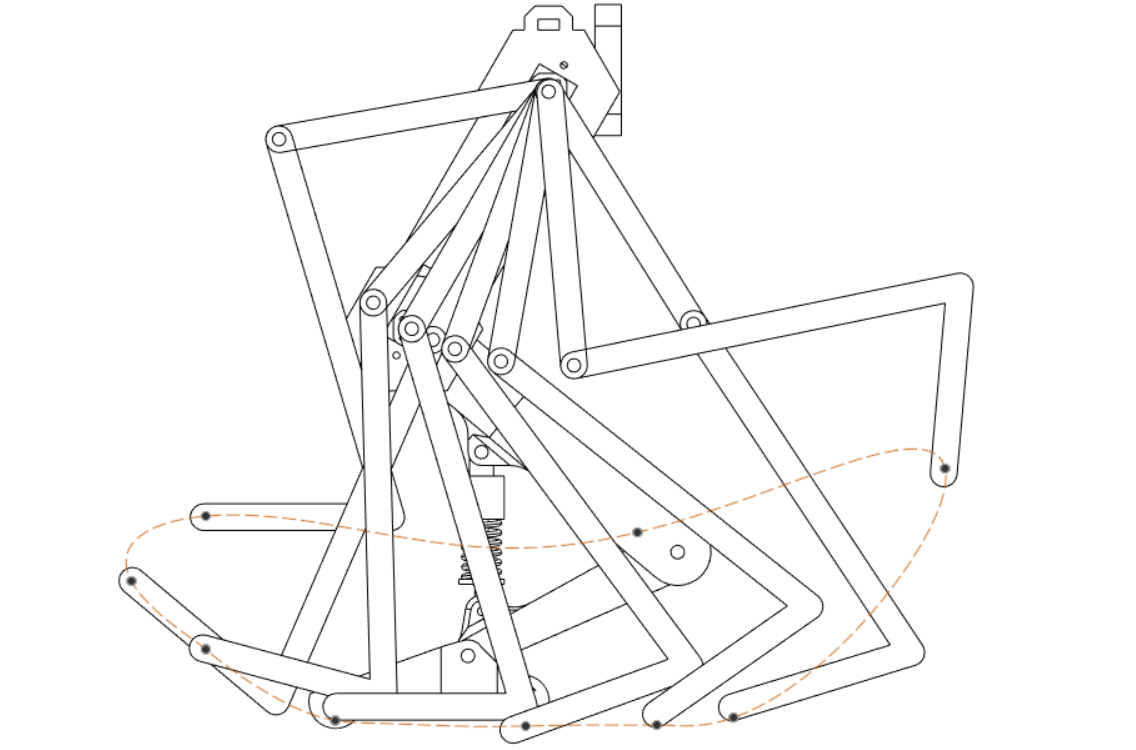
<https://www.megunolink.com/documentation/arduino-libraries/exponential-filter/>

explain the math ryan.



Calculate pitch and yaw n stuff from IMU

<https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/>



“air”

“air”

Contact 1 C2 C3 C4 C5 C6

6 stages are touching ground, 2 are in the air to facilitate the swing forward. The other leg follows the same cycle but is offset by 4 stages, allowing 2 states of ground contact to overlap, fulfilling the double contact state for a total of 26% of the cycle as in figure [bird walk]. It can be broken down into 6 states of ground contact and 2 states of mid-air swing, this is similar for the sideways movement, but is simpler, as the ground contact only requires the moving of the hip servos sideways at a regular speed until the swing phase that swings the leg back to a starting position.

The duration of ther stages can be altered in the code with the time variable attached to each of the sets, but with a single gait cycle broken down into 8 parts, each part comes to 100/8=1205% which is in line with the 13% recorded, this allows a low number of divisions with all the sets having the same duration.

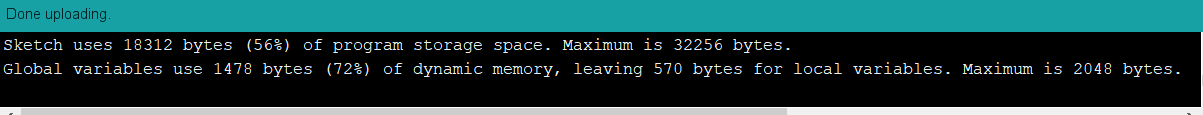
If higher ram with more divisions possible, the time variable can be adjusted to allow the double support phases to come closer to 13% of the cycle.

This is a

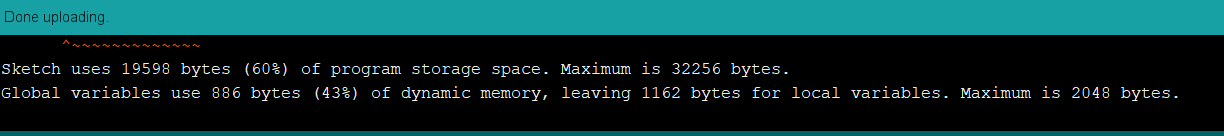
<https://edmond.mpdl.mpg.de/dataset.xhtml?persistentId=doi:10.17617/3.ETFG41>

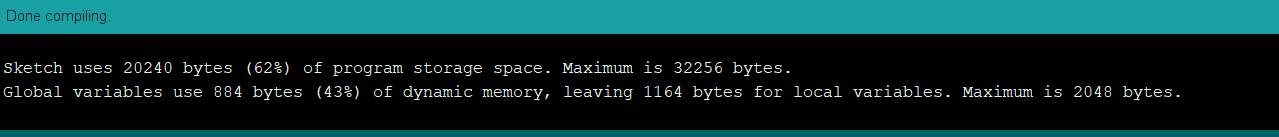
# PROGMEM implementation

Using <https://github.com/Chris--A/PGMWrap> to aid in storing sets in flash memory instead of SRAM to free up space:

Before implementation: 

After:





Allows max divisions to go back up to above 10, allowing for smooth motion and the easing function to actually do things!!!

After this limiting factor is speed of loop execution, as of implementing the PROGMEM, a loop takes on average 0.011 seconds

# Leg planning puppeteering

Empty set was made representing the foot lift, and side to side swing manually tuned to reduce the hip wobbling, counteracting the legs causing the hip to swing back and forth

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Footstep cycle: | % cycle | Empty | Front/back | Left/right |
| 1 | 12.5% | Right leg up + spine counteract | L C3, R air | L C3, R air |
| 1-2 |  | - | R Swing + L walk | R Swing + L walk |
| 2 | 12.5% | Right leg up + spine counteract | L C4, R air | L C4, R air |
| 2-3 |  | Right leg move down | L walk | L walk |
| 3 | 12.5% | Neutral | L C5, R C1 | L C5, R C1 |
| 3-4 |  | - | L and R walk, D support. | L and R walk, D support. |
| 4 | 12.5% | Neutral | L C6, R C2 | L C6, R C2 |
| 4-5 |  | Left leg move up | R walk | R walk |
| 5 | 12.5% | Left leg up + spine counteract | L air, R C3 | L air, R C3 |
| 5-6 |  | - | L Swing + R walk | L Swing + R walk |
| 6 | 12.5% | Left leg up + spine counteract | L air, R C4 | L air, R C4 |
| 6-7 |  | Left leg move down | R walk | R walk |
| 7 | 12.5% | Neutral | L C1, R C5 | L C1, R C5 |
| 7-8 |  | - | L + R walk, D support. | L + R walk, D support. |
| 8 | 12.5% | Neutral | L C2, R C6 | L C2, R C6 |
| 8-1 |  | Right leg move up | L walk | L walk |

Departure from biomimicry,

Due to the low strength of the servos, the gait angles need to be smaller for the servos to be able to actuate to the target positions, and alos due to this reduced dynamic forces, the robot is unable to properly achive foot lift off during its swing phase, essentially maintaining double support throughout its entire gait cycle. With the servos implemented, it is able to shuffle along, but not fully walk like the cycle illustrates, despite while simply increasing the actuation modifiers, such as the angle of gait swings and sufficiently powerful servos would allow for walking as in the above diagram.

This shuffling is due the the compliance of the frame and joints of the robot, when a foot is raised, instead of the foot rising, the entire body will bend to accommodate this new positioning, this compliance is due to the weight of the hip and spine pressing down on the leg assemblies. This causes the overall robot to bend in ways tha3t would not be possible if all parts were rigid. This issue may be alleviated by increasing the motion of the joints, where the body is rigid enough to not allow angular compliance of larger angles, however, the servos are simply not strong enough achieve the torques necessary to perform these larger angled foot steps, regardless of sideways or forwards.

It is the combination of these two issues that limit each other that result in the robot not performing as initially intended. However, knowing this, work was still done to maximise the robot’s ability to maneuver under these new sets of constraints.

In light of this, modifications were made to improve performance in its current form (utilising constant double support. The percentage of time the legs are swinging back forward (supposedly mid-air under normal circumstances) have been reduced to half that of the other stages. With all other parameters the same, this increased walking speed by.

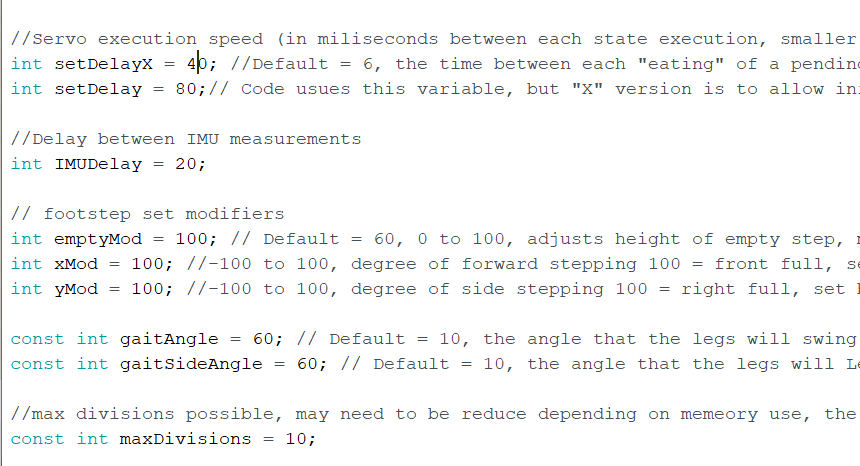
RYAN test to do: original planned motion based on real quail; improvements based on research platform.

SERVOS NOT STRONG ENOUGH TO WALK, CAN SHUFFLE THOUGH

The current movements are broken down into 3 sets that are combined with different weightages, future work may also use this now ealisy expandable framework to break down complex movement into further sub-catagories to further refine and improve control of the CPG. An example would be breaking down the empty set into separate footstep and counterbalance movements, and have the counterbalance be further informed by more IMU data to counteract more dynamic, unexpected perturbances such as potholes, inclined/slippery surfaces, etc.

This system that relies on higher functionality instead of taking into account all the math behind the dynamics, inverse kinematics, etc. not only saves computation but is also much more readily adaptable to new more challenging terrain, where a system whose movements are calculated based on assumptions such as 0-inclined surfaces, no slip, unrealistic perfectly rigid body parts may need to overhaul its entire calculations to take these new variables into account, this system may be altered by simply changing parameters or breaking down the step cycles into more parts as mentioned above.

Demonstration values, slow exaggerated movements.



{50, 24, -20, 0, -64, -94, 5, 0, 14, 10}

Servo 8

Servo 7

Servo 4

Servo 5

Servo 2

Divisions

Servo 9

Servo 6

Servo 3

Servo 1

**9**

**9**

**9**

**9**

**Gait Cycle Stages**

**Rest state**

**1**

**2**

**3**

**8**

**4**

**9**

# Archived debugging code:

// //print pending set

// Serial.println("Pending set:");

// for ( int i = 0; i < maxDivisions+1 ; i++){

// Serial.print("Set indexed at ");

// Serial.print(i);

// Serial.print(": {");

// for ( int s = 0; s < 9 ; s++){

// Serial.print(pendingSet[i][s]);

// Serial.print(", ");

// }

// Serial.println("}");

// }

Goes just before end of respective servo loop in assumePosition():

// //printing temporary column

// Serial.print("tempcolumn[]");

// Serial.print(" for servo number ");

// Serial.print(servoNumber);

// Serial.print(": {");

// for ( int i = 0; i < divisions ; i++){

// Serial.print(tempColumn[i]);

// Serial.print(", ");

// }

// Serial.println("}");

// Serial.print("servo ");

// Serial.print(servoNumber);

// Serial.println("'s transition values calculated");

Goes below respective servo loop in assume position function

//// Printing transitionArray

//Serial.println("Transition array:");

//for (int s=0;s<9;s++){

// Serial.print("colunm array for servo ");

// Serial.print(s);

// Serial.print(": [");

// for (int i = 0; i < divisions ; i++){

// Serial.print(transitionArray[s][i]);

// Serial.print(", ");

// }

// Serial.println("] ");

//}

// //print out new set

// Serial.print("Old set overwritten to new set: ");

// for (int i=0; i<9; i++){

// Serial.print(newSet[i]);

// Serial.print(", ");

// }

// Serial.println(" ");

//Write transitionary array into the pending sets from the back

//Serial.println("Populating pending set:");

//write to pending sets from transition array, transposing matrix

for ( int i = 0; i < divisions ; i++){

for ( int s = 0; s < 9 ; s++){

pendingSet[divisions-i-1][s] = transitionArray[s][i];

//Serial.println(pendingSet[divisions-i-1][s]);

}

//Serial.print("one pending set added at index: ");

//Serial.println(divisions-i-1);

}

**Nomenclature:**

|  |  |  |
| --- | --- | --- |
| m | - | Mass |
| t | - | Time |
|  | - |  |
|  | - |  |
|  | - |  |
|  | - |  |

Set value:

20

0

100

-100

True Servo angle:

159°

109°

134°

139°

Servo Centre - range

Servo Centre + range

Servo Centre

(Centre pre-set at 134°, allowable range pre-set at ±25°)

{{x, x, x, x, x, x, x, x, x, 0},

{x, x, x, x, x, x, x, x, x, 0},

{x, x, x, x, x, x, x, x, x, 0},

{x, x, x, x, x, x, x, x, x, 0},

{x, x, x, x, x, x, x, x, x, 0},

{x, x, x, x, x, x, x, x, x, 0},

{x, x, x, x, x, x, x, x, x, 0},

{x, x, x, x, x, x, x, x, x, 0},

{x, x, x, x, x, x, x, x, x, 0},

⋮

{x, x, x, x, x, x, x, x, x, 0},

⋮

{0, 0, 0, 0, 0, 0, 0, 0, 0, N}}

New position set

Transitionary position sets

Length dependant on maximum divisions allowed by pre-set variable in code

Current position set

Last set containing only “available sets” variable