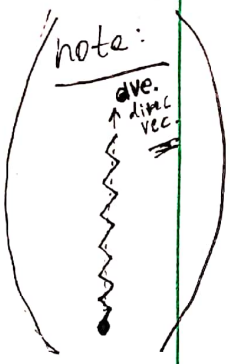
corner inflections

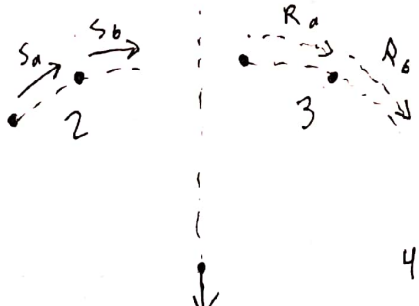
$$\begin{bmatrix} S_a1 & S_b1 & \cdots & R_a1 & R_b1 & \cdots \\ S_a3 & S_b3 & \cdots & R_a3 & R_b3 & \cdots \\ S_a5 & S_b5 & \cdots & R_a5 & R_b5 & \cdots \\ R_a\emptyset & R_b\emptyset & \cdots & S_a\emptyset & S_b\emptyset & \cdots \\ R_a2 & R_b2 & \cdots & S_a2 & S_b2 & \cdots \\ R_a4 & R_b4 & \cdots & S_a4 & S_b4 & \cdots \end{bmatrix}$$

2 DOF leg
can move in arcs w/out issue

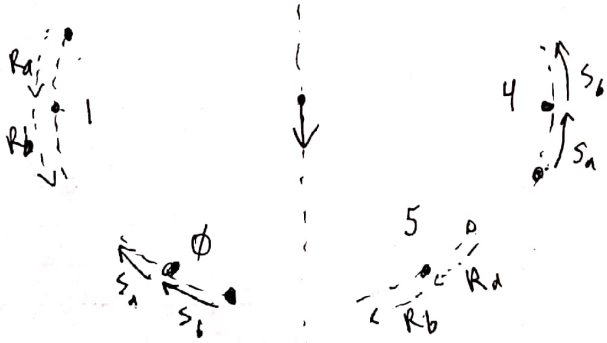
note:



NOTE: two vertices of tripod on same side of motion midline will have same orientation; third vertex opposite orientation

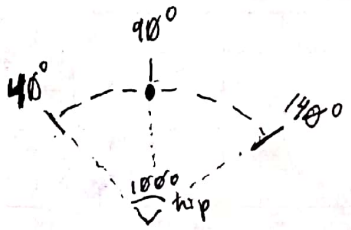


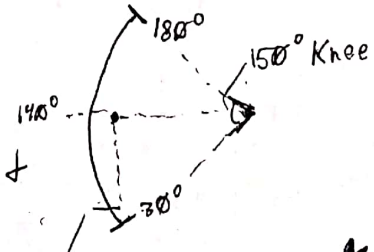
NOTE:
servos wiggle, 5° intervals for reliable differentiation
⇓
20 hip angles, 30 knee angles



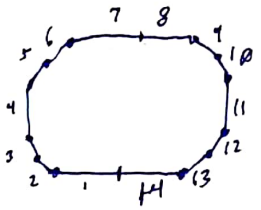
6ØØ combinations/ ee-positions per leg

recall:





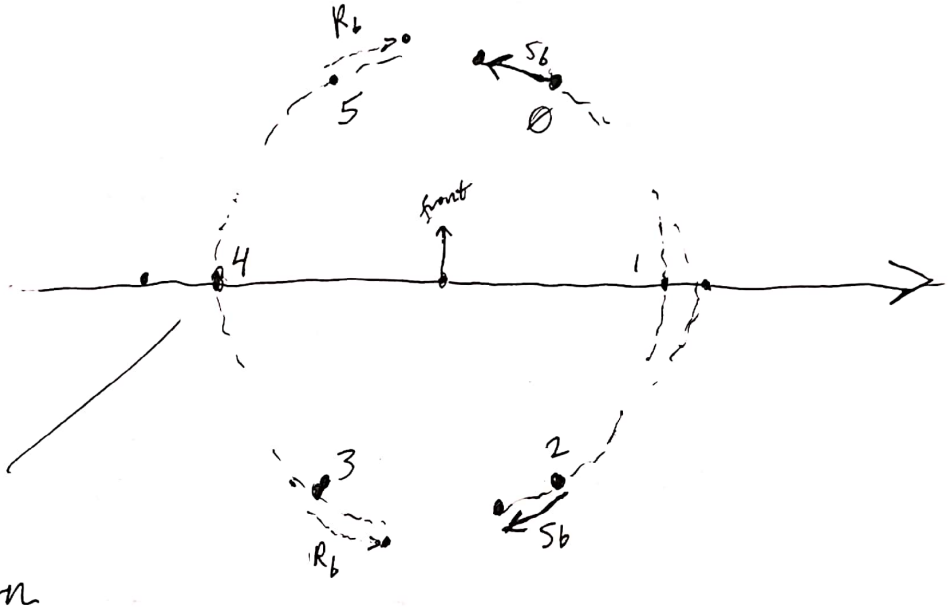
(reliably in stance at 6Ø°) ★



$\emptyset + 6 + 2$

$1 + 6 + 2$

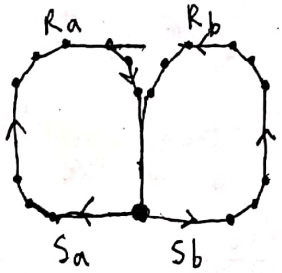
tripod 1, knee 9



could use a 'figure-8' pattern to address :

e.g.

w/ opposite in front leg - on direc axis



$$\begin{bmatrix} S_a 4 \cdots R_a 4 \cdots S_b 4 \cdots R_b 4 \\ R_a \blacksquare \cdots S_a 1 \cdots R_b 1 \cdots S_b 1 \end{bmatrix}$$

inflection points

if buttonRB = 1:

- turn right
  - turn_config
  - +1 direction
  - return to stance when buttonRB = 0

if elevbutton1 = 1

riseheight = high

riseheight = norm when elevbutton1 = 0

if buttonLB = 1:

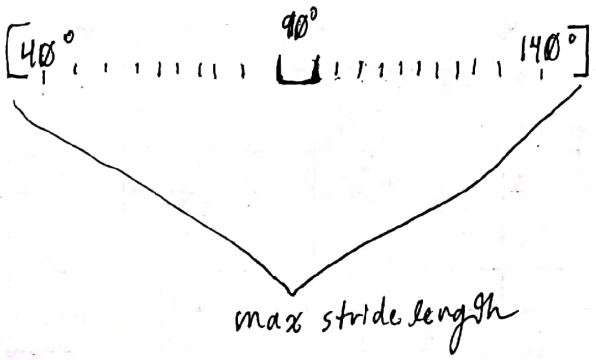
- turn left
  - turn_config
  - -1 direction
  - return to stance when buttonLB = 0

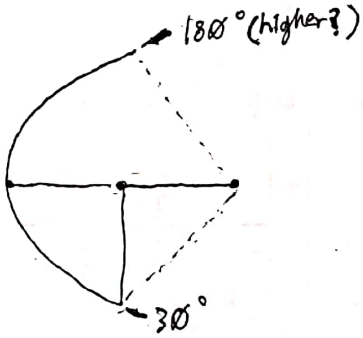
8 paths
0° N
30° NNE
90° E
150° SSE
180° S
210° SSW
270° W
330° NNE

if joystickaxis 1 != 0 & joystickaxis 2 != 0:

- get direction and magnitude of combined axes 1, 2 (jsdirec, jsmagn)
  - fit direction to closest hard-coded direction
  - fit magnitude to point on stride length allow. values
- if js magn. changes: (changes enough to change allow. values...)
  - switch stride length on next return to mid stance pos. for each leg

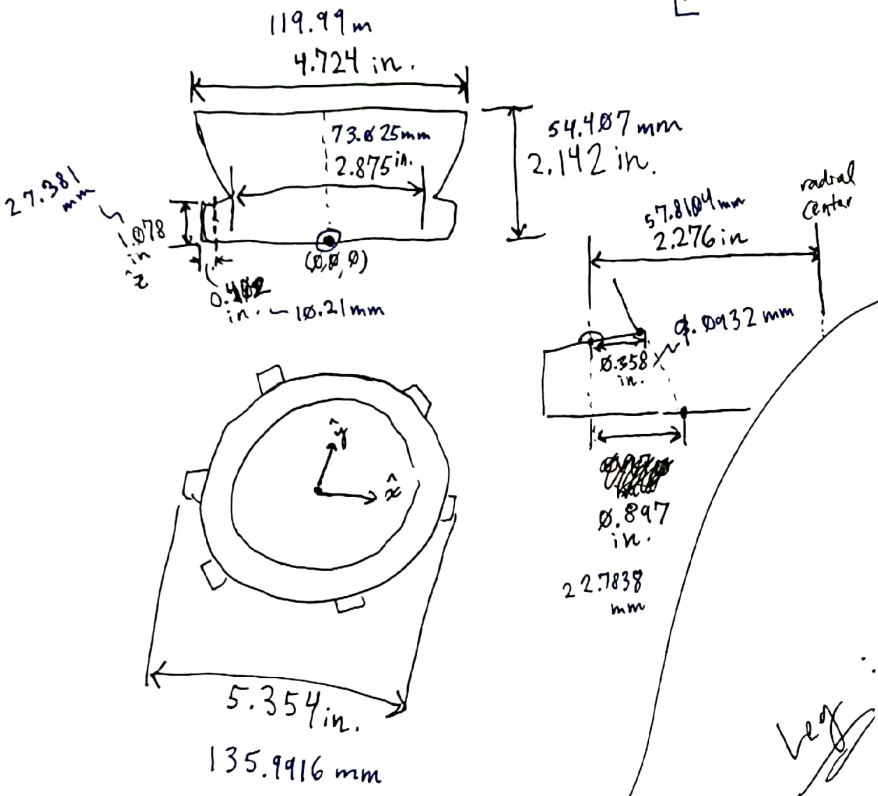
- if js direc. changes:
  - return legs to mid stance pos and start anew





- js magn. goes to 255 ⇒ 20 different stride angles
  10 mirrored angles → change stride length for every ~25.5 js magn.

base:



1.058in COM

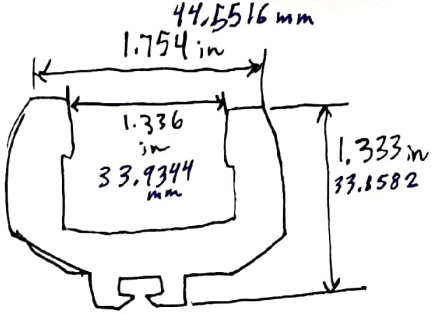
$$\begin{bmatrix} 0.00086861\,m \\ -0.00129677\,m \\ 0.0233\,0199\,m \end{bmatrix}$$

single COM

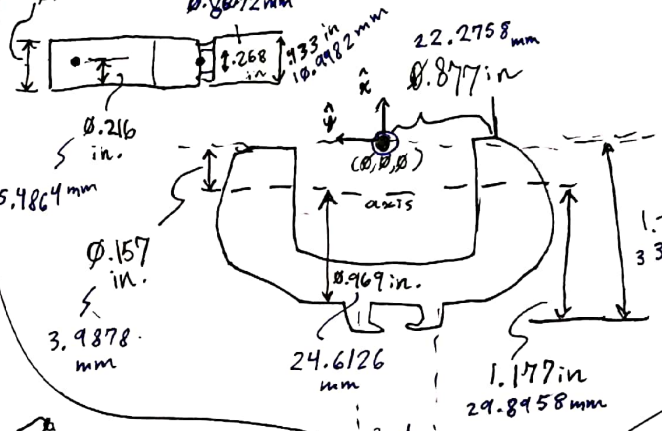
$$\begin{bmatrix} -0.018167\,05m \\ -0.00103767m \\ 0.00551701m \end{bmatrix}$$

joined COM

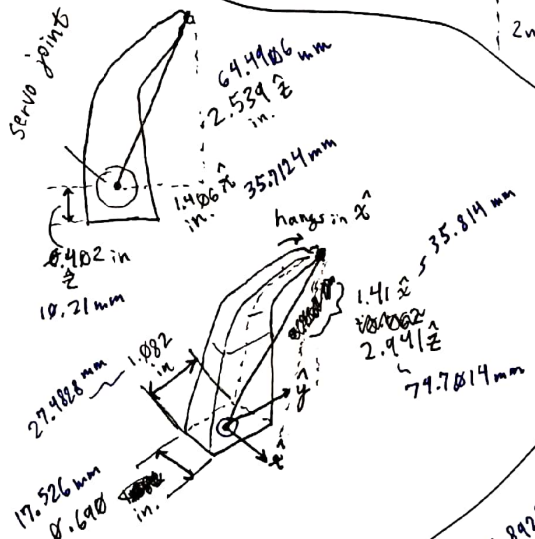
$$\begin{bmatrix} -0.02861993m \\ -0.00082572m \\ 0.00538506m \end{bmatrix}$$



1.754
.877
1.677
.0+77
1.74



Knee joint



Leg:

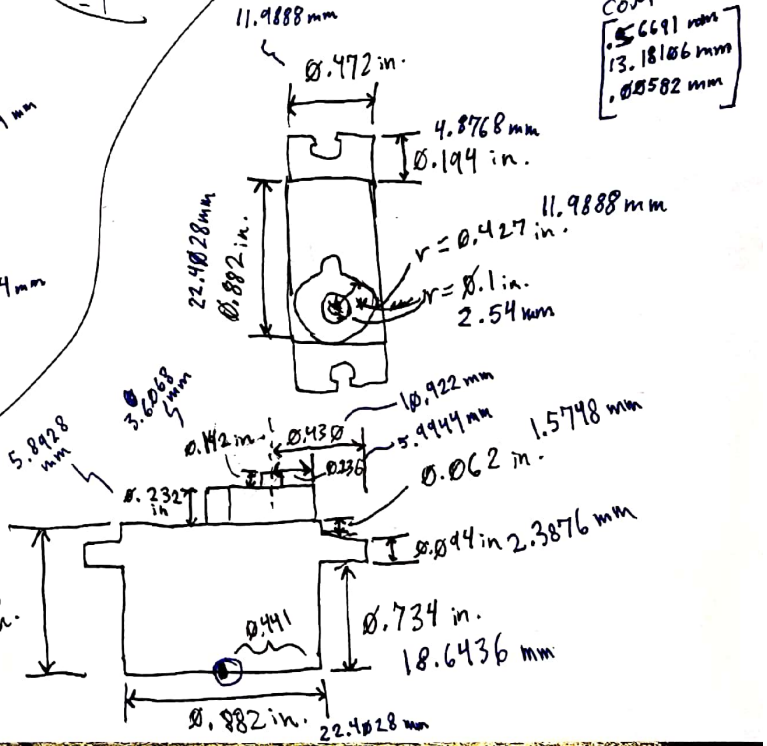
COM

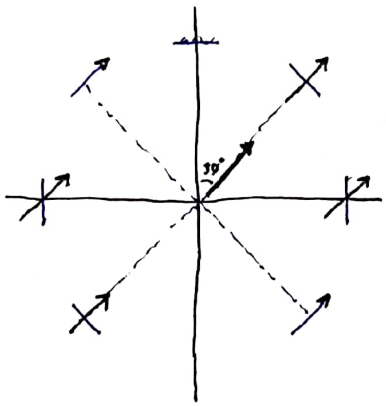
$$\begin{bmatrix} 0.00355399\,m \\ -0.00068646\,m \\ 0.03353051\,m \end{bmatrix}$$

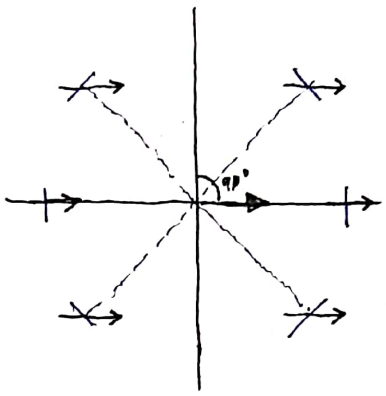
Servo MG995

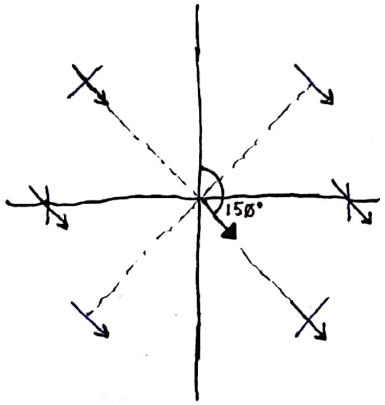
COM

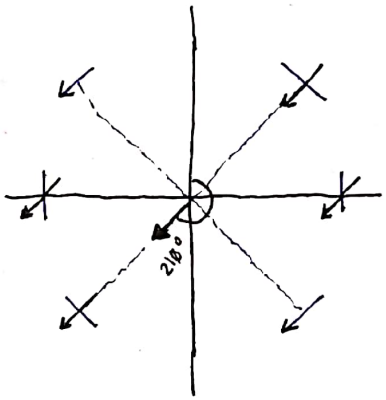
$$\begin{bmatrix} .5\,6691\,mm \\ 13.18106\,mm \\ .00582\,mm \end{bmatrix}$$

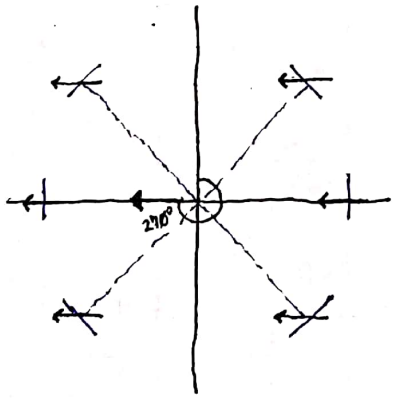


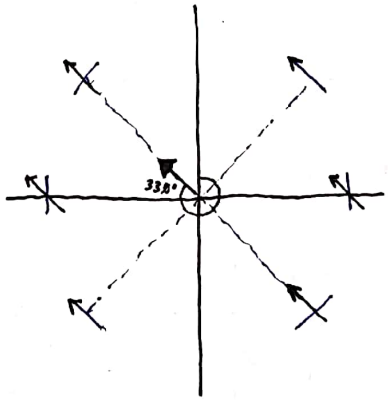




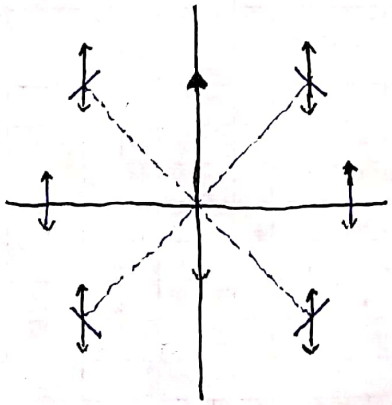








NOTE:
legs ±30° off midline,

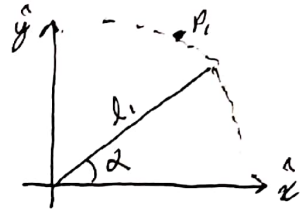


0°/180°

3 Common Gaits:
(1) Metachronal
(2) Ripple
(3) Tripod

IK:

$$\alpha = \arctan\left(\frac{y_i}{x_i}\right)$$



(1) $P_i^{t+1} = \left(P_i^t - \frac{v}{f}\right) \cdot R_z\left(\frac{\theta}{f}\right)$ (2) $P_i^{t+1} = P_i^t - \frac{v}{f}$

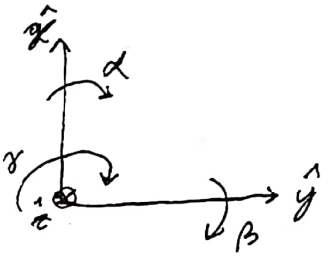
(1): Calculate next pos. w/ rotation

(2): Calculate next position

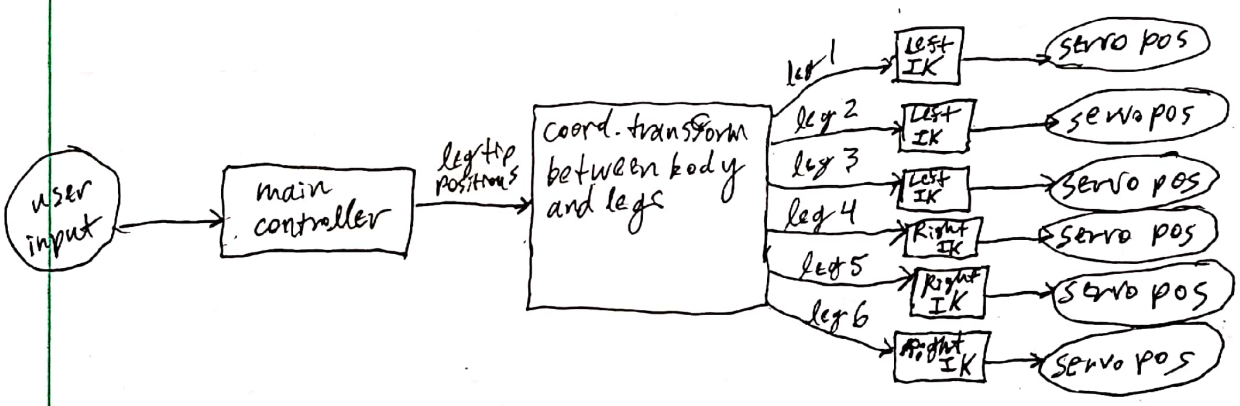
terrain handling:

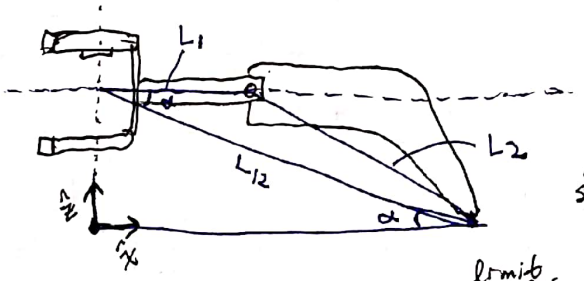
$\Delta h = a \cdot \tan(\alpha)$ → extend to $a \cdot \tan^{pitch}(\beta)$, $a \cdot \tan(\gamma)$ yaw

α: ground to body angle



note: this paper uses elliptical motion





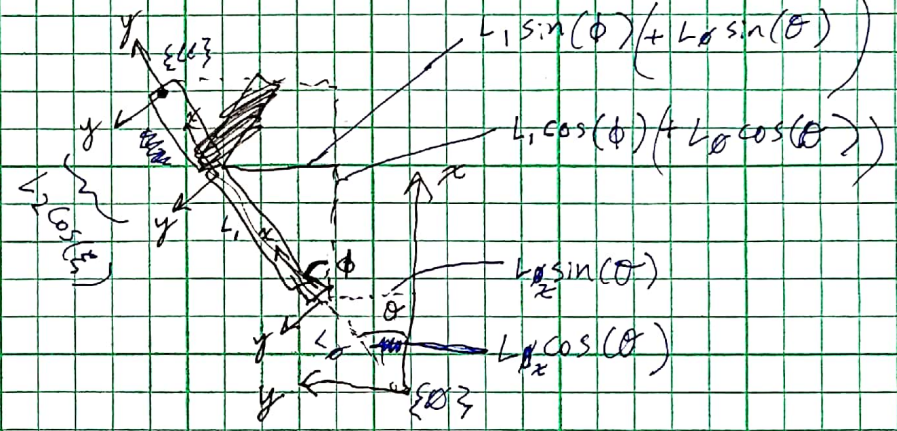
Sample user input
↓
sample IMU input
↓
Calc. height & default leg pos
↓
create a leg queue
↓
check buttons for mode change (this can allow gait transitions)
↓
alter leg pos for height
↓
determine if addt. legs can be lifted
↓
precalculate traj.(s) for legs to be lifted
↓
output next pos of legs in swing phase based on precalc. trajectories
↓
calc. and output next position for leg in stand phase
↓
translate leg positions to coord. system for each leg IK
↓
IK calculate servo pos(s) based on provided leg positions
↓
limit servo pos due to hard constraints
↓
output servo pos(s)

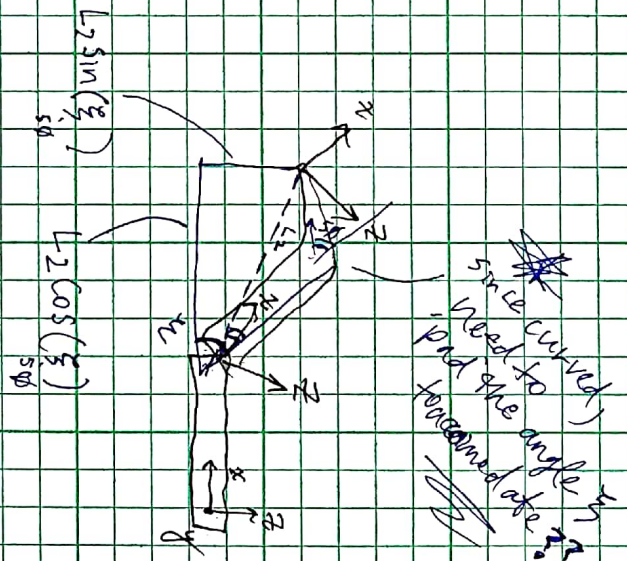
SOH CAH TOA

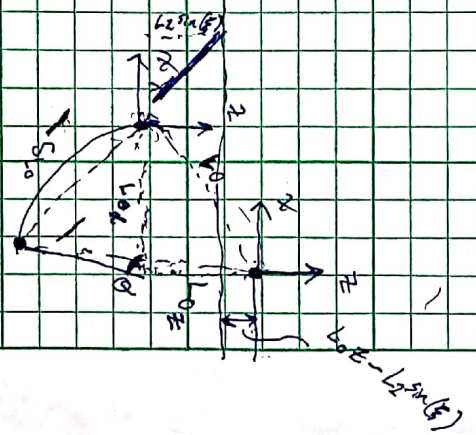
$$arcz = L_2 \sin\left(\xi_{Sθ}\right) + L_{0z} - L_{0z}$$

$$arcy = L_{0x}\sin(\theta) + \left(L_1 + L_2\cos\left(\xi_{Sθ}\right)\right)\sin(\phi)$$

$$arcx = L_2\cos\left(\xi_{Sθ}\right) + L_1\cos(\theta) + L_{0x}\cos(\theta)$$







whole corner length

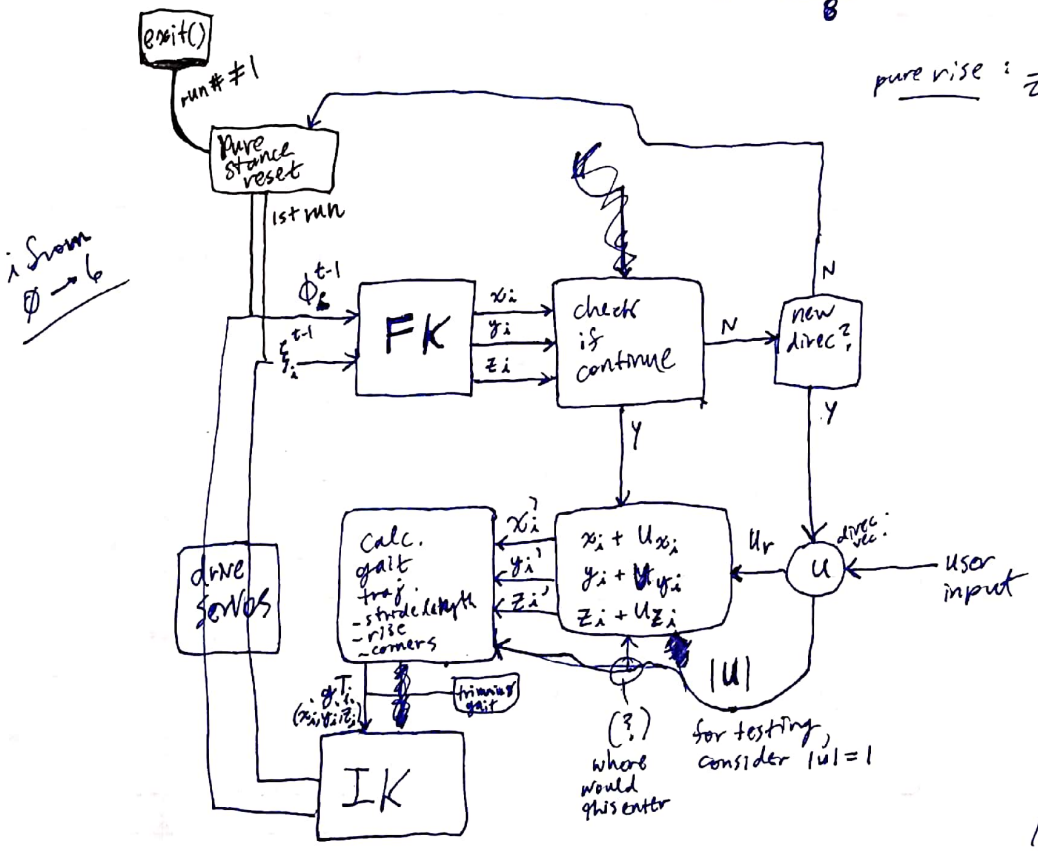
$$\text{stride length} = \frac{u_r}{|u|} - \frac{\frac{u_r}{|u|}}{8}$$

corner increments: $\frac{u_r}{|u|}$

x_i, y_i: $\frac{}{8 \cdot 2}$

z: $\frac{\text{rise height}}{8}$

pure rise: z: $\frac{\text{rise height}}{8} \cdot 2$



i from $\phi \to 6$

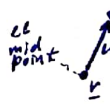
NOTE: will have to shift ϕ_i, ξ_i for driving servos; i.e. $0° \Rightarrow 90°$

$5 \cdot 14 = 70$ entries in $g_i T$ (140)

$$g_\phi T = \begin{bmatrix} p1 \\ p2 \\ p3 \\ \vdots \\ p13 \\ p14 \end{bmatrix} = \begin{bmatrix} (\phi_1, \xi_1) \\ (\phi_2, \xi_2) \\ \vdots \\ (\phi_{13}, \xi_{13}) \\ (\phi_{14}, \xi_{14}) \end{bmatrix}$$

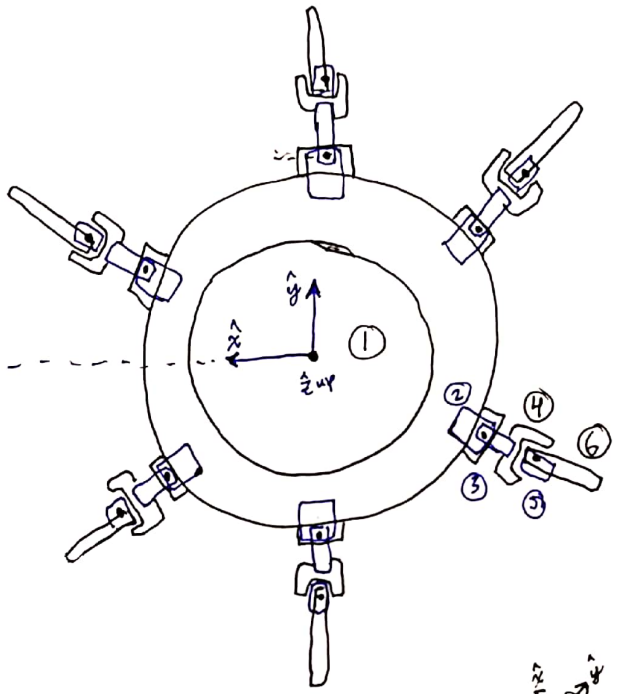
$$g_i T = \begin{bmatrix} g_\phi T \\ g_1 T \\ \vdots \\ g_5 T \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} (,) \\ \{,\} \\ (,) \end{bmatrix} \\ \vdots \\ \begin{bmatrix} (,) \\ \vdots \\ (,) \end{bmatrix} \end{bmatrix}$$

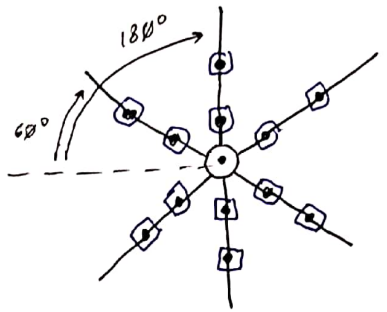
$\underline{r}' = \underline{r} + u_r$



divide u_r by $|u|$ to normalize for tests?

θ_i = ~~servo #~~, $i \to 11$





□ = servo w/ mass and no DOF on one side, IR on other

base
center of mass
↓ fixed
servo COM 1
↓ fixed
servo horn 1
↓ variable
Knee CoM
↓ fixed
servo horn 2
↓ Variable
servo COM 2
↓ fixed
leg CoM
↓ fixed
foot

×6

NOTE: measurements at all Servos = 90° best to establish transformations

↳ NOTE: legs will still be bent due to curvature in foreleg

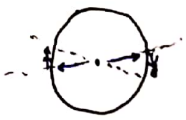
hip servos ($0 \to 5$): $T_{\theta_{i_{0-5}}} = \begin{bmatrix} \cos(\theta_i) & \sin(\theta_i) & 0 & x_i \\ -\sin(\theta_i) & \cos(\theta_i) & 0 & y_i \\ 0 & 0 & 1 & z_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$

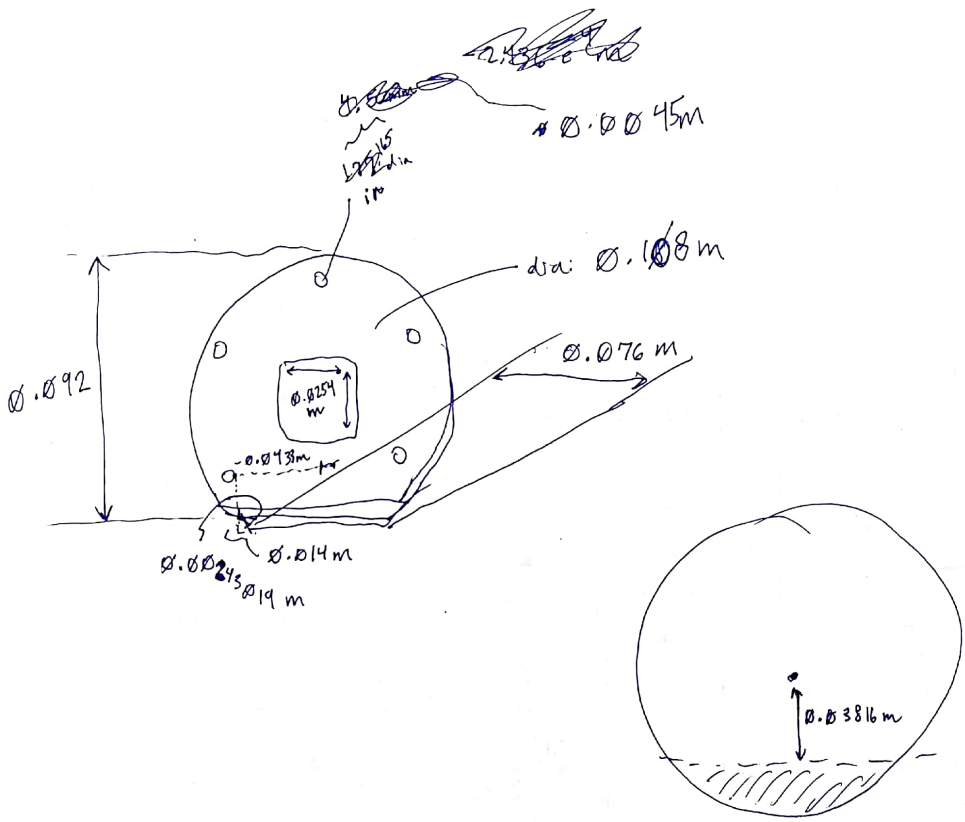
Knee servos ($6 \to 11$): $T_{\theta_{i_{6-11}}} = \begin{bmatrix} \cos(\theta_i) & 0 & \sin(\theta_i) & x_i \\ 0 & 1 & 0 & y_i \\ -\sin(\theta_i) & 0 & \cos(\theta_i) & z_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$

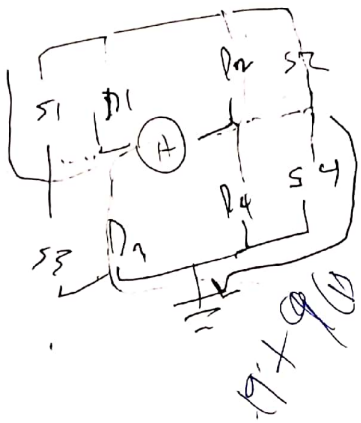
NOTE: legs not perfectly symm.

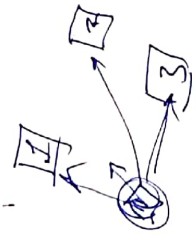
$\hat{y}$: -0.00277272

$\hat{x}$: -0.04216314




Ø.0045m
dia: Ø.108m
Ø.092
Ø.076 m
Ø.0254 m
Ø.0473m
Ø.014m
Ø.0024
Ø19 m
Ø.0316m





(n = 17, [∅ : 16])

ps3joy.

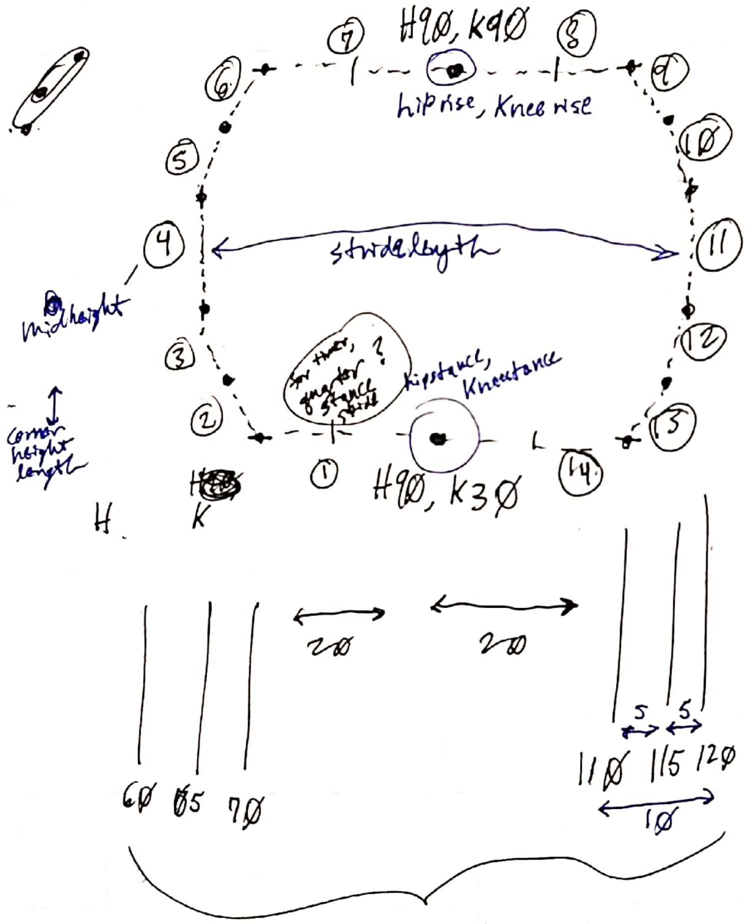
[select R2 R3 start L2 R2 L1 R1 △ ○ × □ PS
∅ , ∅ , ∅ , ∅ , ∅ , ∅ , ∅ , ∅ , ∅ , ∅ , ∅ , ∅ , ∅ , ∅ , ∅ , ∅ , ∅ , ∅]

[maintain history w/ git transfer
- sep. branch, commit ; delete everything else
- add as a remote to new repo, pull

Hexapod notes:

Hips: 90°
Knee: 25° → lowest supported w/out overdriving servos ↗ all six legs on ground
↘ also w/ 3 legs on ground

↓
30° provides safer, more reliable range } NOTE: 35° becomes somewhat unstable (foot grip?)



90 — 85 — 80 — 40 — 40 — 35 — 30

Safe values for avoiding collision; not optimized

14 commands, 7 front, 7 backs

$$\left(\frac{\frac{\text{stride length}}{2}}{3}\right) = \text{trisect}$$

$$\frac{\text{trisect}}{2} = \tfrac{1}{2}\text{corner}$$

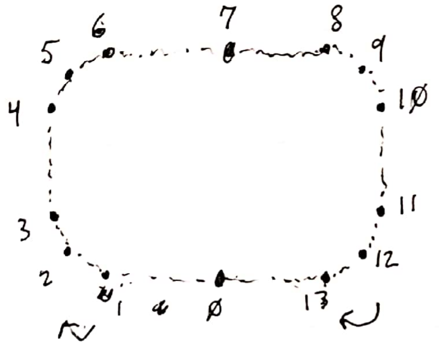
$$\text{trisect} \cdot 2 = \tfrac{2}{3}\text{stance stride}$$

int
a1 =
a2 =
a3 =

while: fn(a i+1)

fn(int(str('a'+i)))

[↓ . . .]



Tri 0: 0, 1, x, x, x, x, 2, 3, 4, 5, 6, 7

Tri 1: 7, 8, 9, 10, 11, 12, 13, 0, 1,

fairly stable dependent on wiggle in joints

4 hold for half stride — 9 for full stride

T0: 0 1 X X X X 2 [3] [4] 5 6 7 8 9 10 11 12 13 [0] [1] X X X X X X X X 2

T1: 7 8 9 10 11 12 13 [0] [1] X X X X X X X X 2 [3] [4] 5 6 7 8 9 10 11 12 13 0

8 hold for full stride

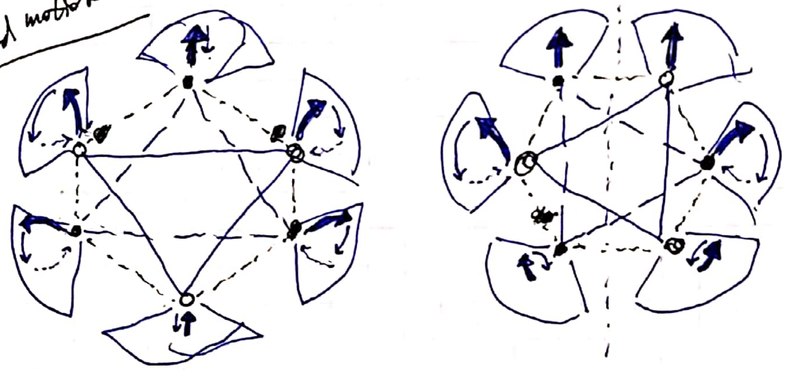
5 hold for half stride

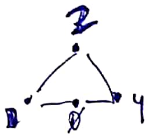
T0: 0 1 X X X X X 2 [3] 4 5 6 7 8 9 10 11 12 13 0 [1] X X X X X X X X X X

T1: 7 8 9 10 11 12 13 0 [1] X X X X X X X X X X 2 [3] 4 5 6 7 8 9 10 11 12 13

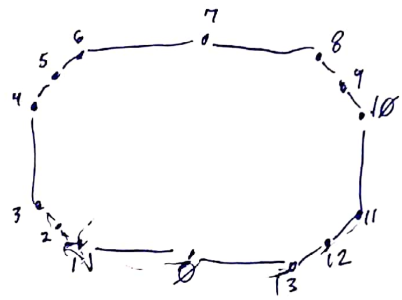
10 hold for half stride

Ex forward/backward motion:





Ø 1 2 3
2 3 Ø 1

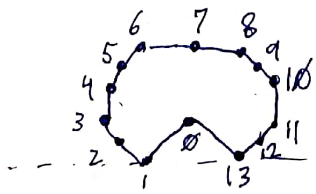


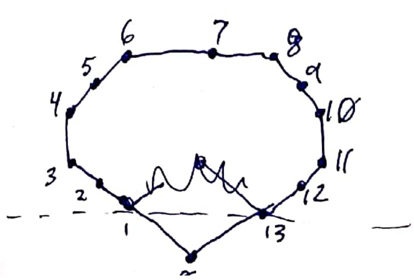
Ø

13 | Ø 1 1 1 1 1 2 3 | 4 5 6 7 8 9 10 11 12 | 13
6 | 7 8 9 10 11 12 13 Ø | 1 1 1 1 1 1 1 1 1 | 2

5

Ø 1 1 1 1 1 2 3 4 5 6 7 8 9 10 11 12 13 Ø
7 8 9 10 11 12 13 Ø 1 1 1 1 1 1 1 1 1 2





rosrun joy joy_node

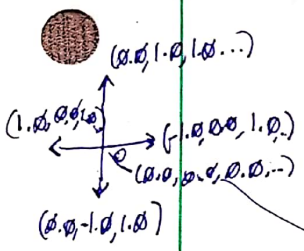
"rostopic echo —"

ROS topic /joy

- stamp:
  - time stamp
  - nsecs
- frame_id: '' ☆ (?)
- axes: [0.0, 0.0, -0.0, 1.0, -0.0, 0.0, 1.0] (L stick, R stick; 17 zeros, 0 or 1)
- buttons: [0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,0] (no input values (default))
  × ○ △ □ LB RB LT RT sel start PS L3 R3 (L1) (R1) (L2) (R2)

of course, roscore must be running :)

rostopic type /joy
⇒ sensor_msgs/Joy



term that is 0.0 will go from 0.0 → 1.0 to provide between those base angles

---

topics: /joy joy: controller input

(screenshot: !!!! joy on desktop)

! • downloaded python3-~~pkg~~ catkin_pkg
↳ rosdep uninstalled...?

NOTE:
does ~~joystick~~ have some amplitude measure

proposal:
leading 1.0 in each ~~base~~ angle will scale 0.0 → 1.0 for magnitude

① roscore
② rosparam load ~~joystick~~ joy_param.yaml
③ rosrun joy joy_node

(assumes, but how?)

VS. ② rosparam set joy_node/dev "/dev/input/js0"

Note: catkin clean hexapod Motion interrupted!

Set Up Access to SMBus Resources

First point: in most OS level interactions, the I²C bus is referred to as SMBus. Thus we get our first lines of code. This imports the smbus module, creates an object of type `SMBus`, and attaches it to bus "1" of the Pi's various SMBuses.

```
import smbus
bus = smbus.SMBus(1)
```

We have to tell the program the part's address. By default, it is **0x40**, so set a variable to that for later use.

```
addr = 0x40
```

Next, we want to enable the PWM chip and tell it to automatically increment addresses after a write (that lets us do single-operation multi-byte writes).

```
bus.write_byte_data(addr, 0, 0x20)
bus.write_byte_data(addr, 0xfe, 0x1e)
```

Write Values to the PWM Registers

That's all the setup that needs to be done. From here on out, we can write data to the PWM chip and expect to have it respond. Here's an example.

```
bus.write_word_data(addr, 0x06, 0)
bus.write_word_data(addr, 0x08, 1250)
```

The first write is to the "start time" register for channel 0. By default, the PWM frequency of the chip is **200Hz**, or one pulse every 5ms. The start time register determines when the pulse goes high in the 5ms cycle. All channels are synchronized to that cycle. Generally, this should be written to `0`.

The second write is to the "stop time" register, and it controls when the pulse should go low. The range for this value is from `0` to `4095`, and each count represents one slice of that 5ms period (5ms/4095), or about 1.2us. Thus, the value of 1250 written above represents about 1.5ms of high time per 5ms period.

Servo motors get their control signal from that pulse width. Generally speaking, a pulse width of 1.5ms yields a "neutral" position, halfway between the extremes of the motor's range. 1.0ms yields approximately 90 degrees off center, and 2.0ms yields -90 degrees off center. In practice, those values may be slightly more or less than 90 degrees, and the motor may be capable of slightly more or less than 90 degrees of motion in either direction.