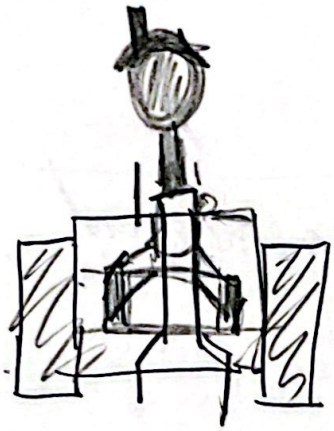
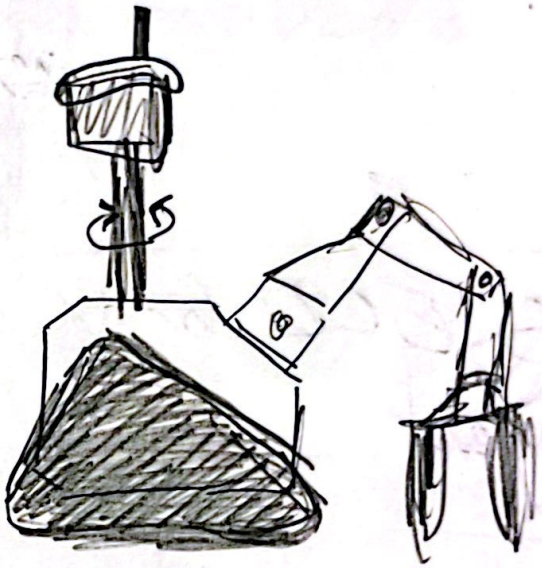
Ik behaving strangely, better than before though

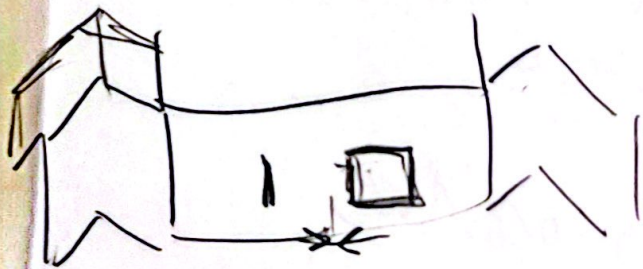
- visualize URDF with applied joint states to see what it thinks is happening

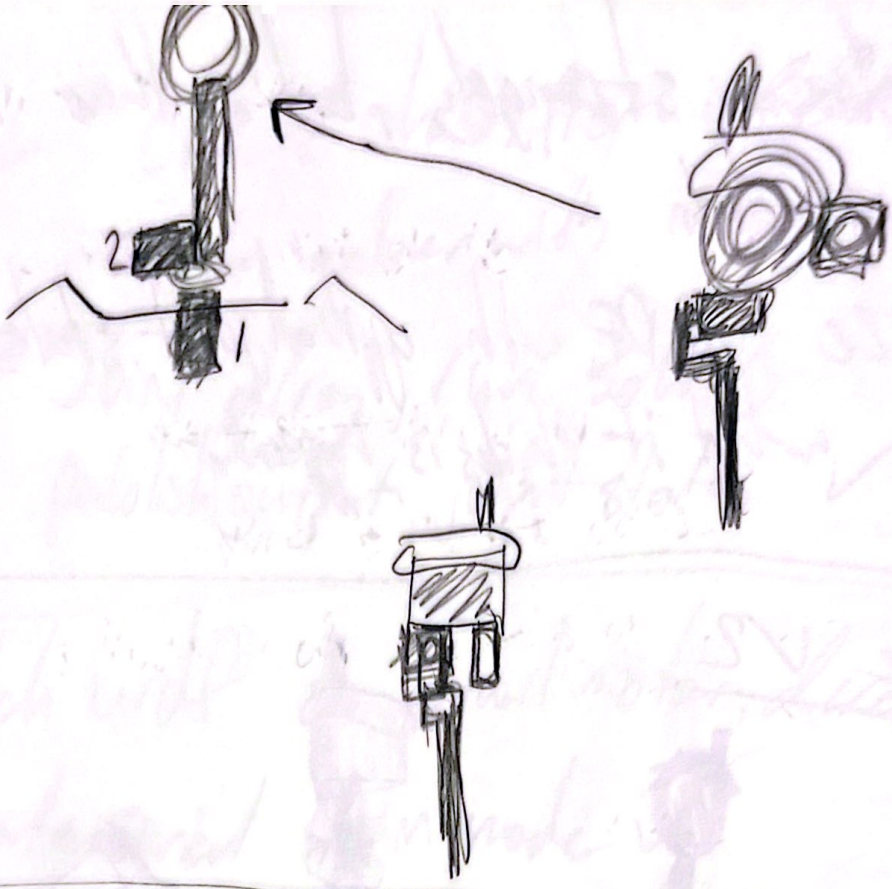
---

Frame V2









AWS EC2 for VLA training and deployment

vast.ai - cheap but not AWS integrated

AWS - reliable but expensive

- Trainium - hard to use, efficient
- g6e - good range of RAM + price
- g4n - 16gb gpu, could be good for inference @ $26

Vast.ai - H100 not that expensive
A100 even cheaper

Train on Vast, deploy on AWS

AWS g4dn - 16GB RAM (T4 gpu)
g6e - 24 GB RAM (L40 gpu)

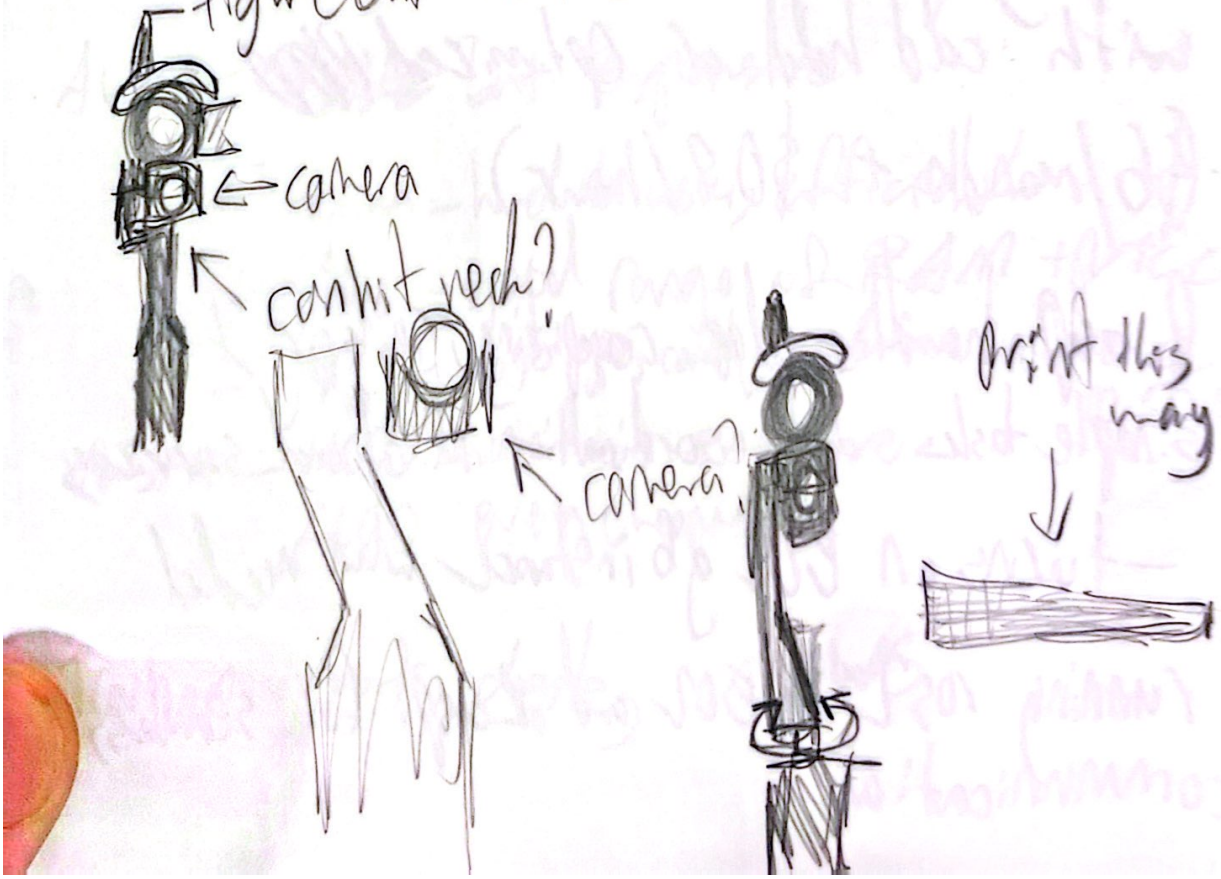
- only host AWS instance when need access to VLA model.
  - how to determine when VLA is needed?

Current workflow

1. Fine tune using vast.ai H100 ($1.5/hr)
2. download fine tuned model to lightsail
3. deploy model on AWS G6 instance with cold hold or optimized HDD ($6/month + $0.8/hour)
4. Raspi handles edge computing, sensor IO, simple tasks and coordination of cloud services
   - turns on EC2 g6 instance when needed
5. running ros2 on EC2 and Raspi for seamless communication

Current blockers

- Electronics not done
  - ~~arm cables~~ / detailed cable on servo board
  - ~~motor pdb~~? (resolder)
  - Raspi pdb + sensor pdb
  - encoder wheel
  - IMU wiring
  - camera wiring
  - face screen wiring
  - second servo board
- Mechanical not done
  - mount tracks
  - print new neck
  - print camera mount
  - figure out head servo movement



Software blockers

- inverse kinematic solver (5 dof)
  - well defined, can be found analytically (openrave docker?)
- Ros2 control setup
  - command → motors
  - encoders → topic (rate?)
  - imu → topic
  - camera → topic

---

once all these blockers are done, can move on to creating intelligent control systems:

- VLA control
- higher level LLM decision making
- lower level behavior tree decision making (charging, handing off control to diffent models)
- memory system?
- online fine tuning?

pwm wiring

14, 16, 17, 18, 19, ~~25~~, 26, 27, used

remaining: ~~24~~, 13, 21, ~~22, 23~~, 32, 33

remaining outputs: 4 × motors/pwm
2/3 servos

motor pins:
12, 15, 22, 23

32, 33, 13 head servo pwm

more servos? Lazy..... not critical to performance....

Zima planning:

what is needed to make him work?

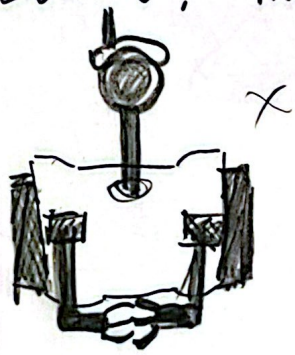
Mechanical:

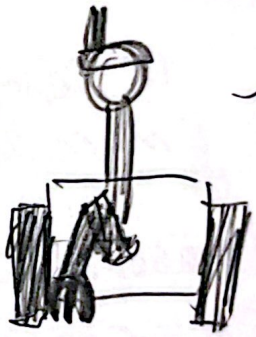
Frame - holds all components, but should look good

Tracks - taken from toy

arm - needs to be able to grip clothes, other basic objects

2 arms? not feasible size wise





Electrical -

battery - overspecced

dc motors - may be too weak?
gear reduction?

Servos - overspeced, but large

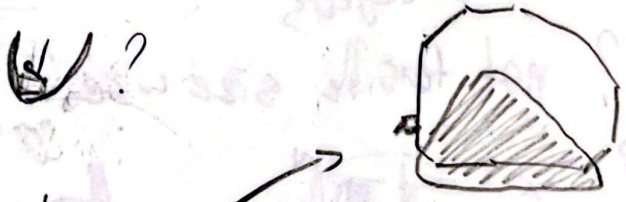
Electrical:

Raspi 5 - Current draw not an issue

Esp32 - lots of connections, may be a good idea to get female header pins for board connections

IR sensors: need to be modified still

Roller switches -
in arm palm? on rear?



Ultrasonic sensors:
similar to ir, but could be used for 2d lidar mapping?

camera could perform similar?

ultrasonic + camera?
can pinpoint depth estimate

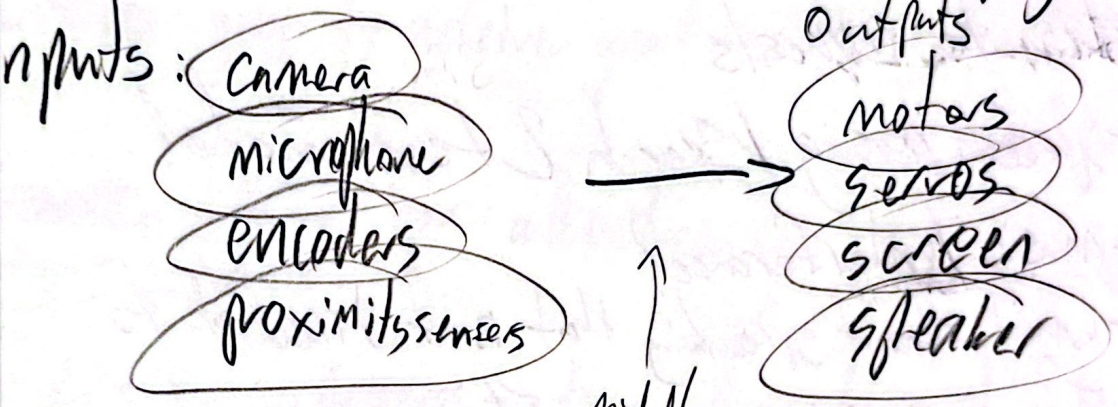
thare:

LLM - local vs online api

SLAM - algorithm performance over resp: could be slow

State estimation: motor encoders + IMU

planning - complex behavior tree?
goap, goal oriented action programming



system overview:

- needs to stably control movement
- needs to keep track of state (position)
- needs to have "emotional" behaviors
- needs to accurately grasp objects
- needs to dynamically respond to requests
-

System requirements / applications

- fetch objects
- clean floor
- hold basic conversation
- charge itself

} keep it simple

fetch object / clean floor requirements

- Listen to requests
  - speech parsing / speech 2 text model
  - music / interference?
  - what about speaking that isn't directed to it?
  - "hey zima" command start like hey google?
- Recognize request
  - pass spoken request to LLM
- Determine current goal based off of request.
  - request importance, queue more important goals first

- utilize goap to achieve goal

Hold basic conversation requirements

- Listen to requests, same as previous
- Pass spoken words to LLM
- determine current goal (respond)
- utilize goap to achieve goal

Same pipeline for all actions?

Listen → process → think of goal → plan → create

utilize LLM for goap planning?
or more optimization based?

Memory capabilities?
Saving important actions, conversations, etc as memory files?
"what did you do yesterday" → "I cleaned, fetched a sock, etc"

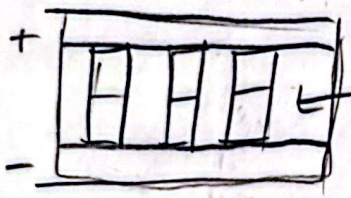
LLM of fine tuning?

VLA / end-to-end model

for arm, remainder of robot can remain classical?

MuJoCo sim for training

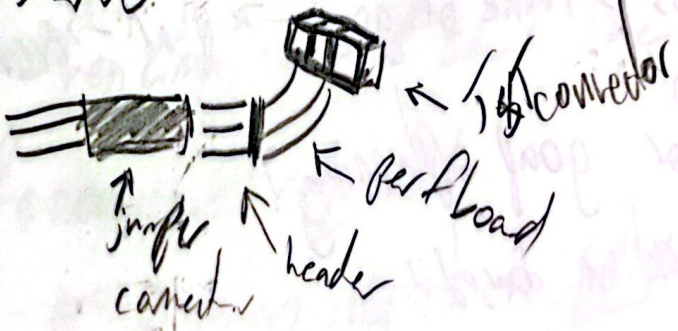
PDB

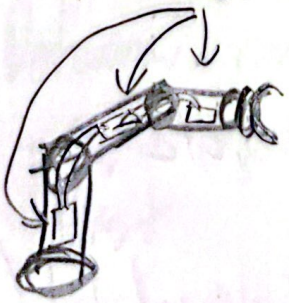


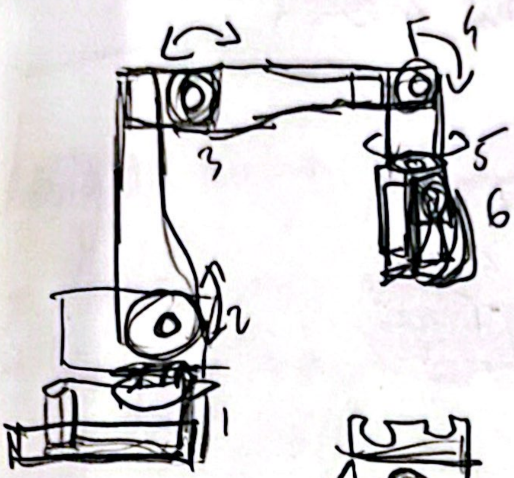
regular connectors?

↑ visuomotor end to end coded VLA too complex

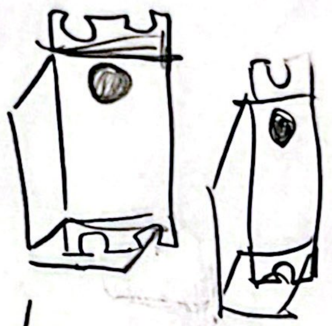
Servo

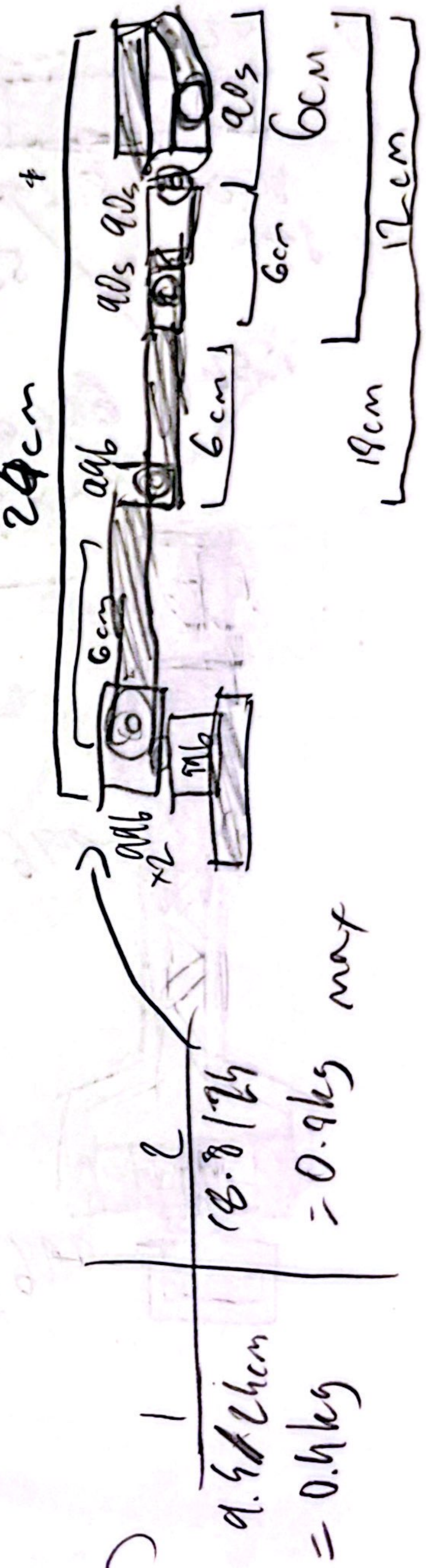






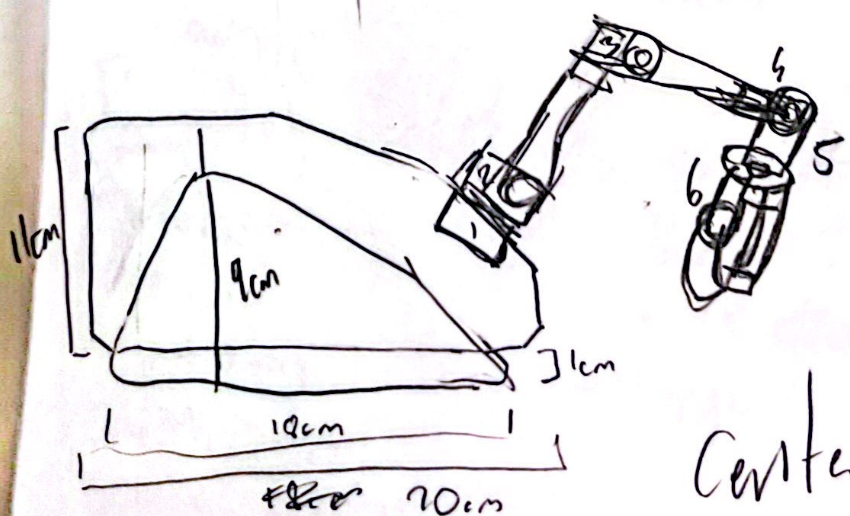
6DOF





physics calcs:

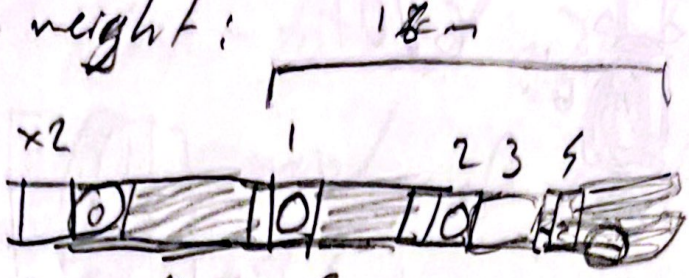
Mg996r:	Mg90s:
max torque 1.4 kg/cm	max torque 1.8kg cm



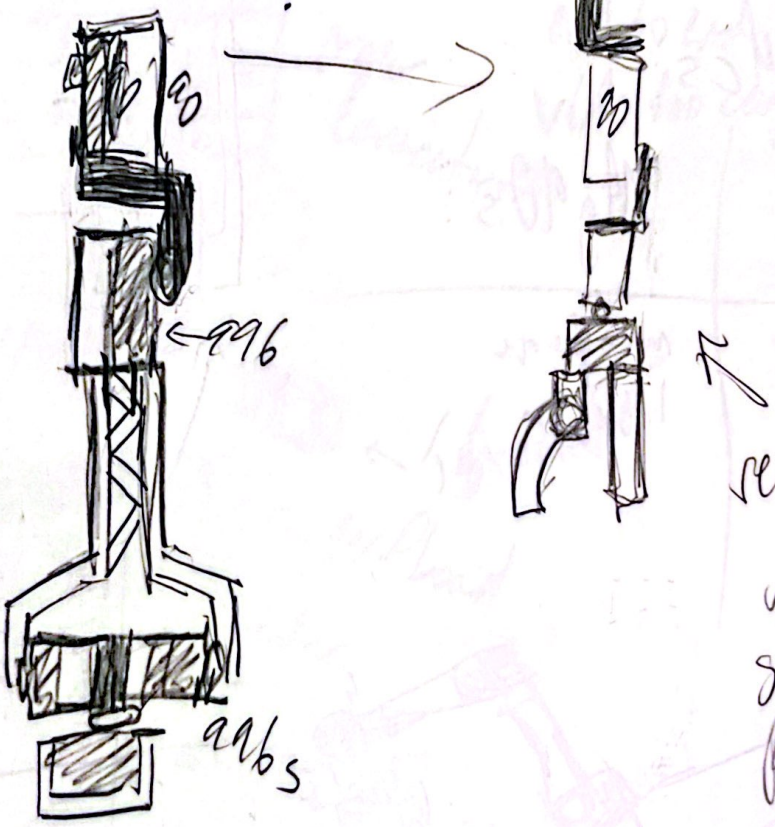
Center of gravity concerns?

2 motor setup : ~800 gram max

arm weight:



issue 600 gram max?

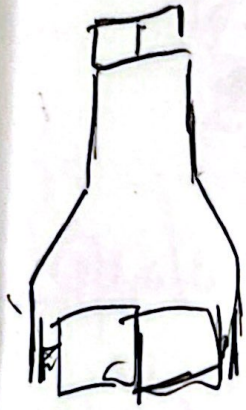


replace with string pully mechanism to reduce weight?

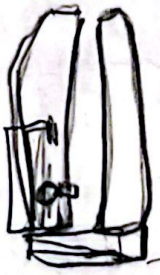
Arm modelling plan:

250 8972295

- design longer shoulder hinge
  - double motor mount
  - servo attachment pattern ✓
    - test print
- design shoulder to elbow connection ✓

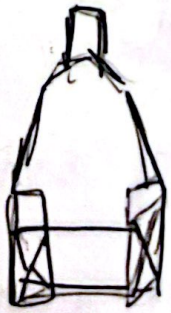


- design elbow to wrist connection ✓
- wrist to hand ✓
- hand to fingers ✓



- assemble full arm ✓

- fix shoulder to elbow ✓
  - thinner channels ✓
  - more clearance with edges ✓
  - longer for 140° clearance ✓



Print finger from tpu ✓

assemble full arm ✓

test each motor individually ← fail ✓

create pdb

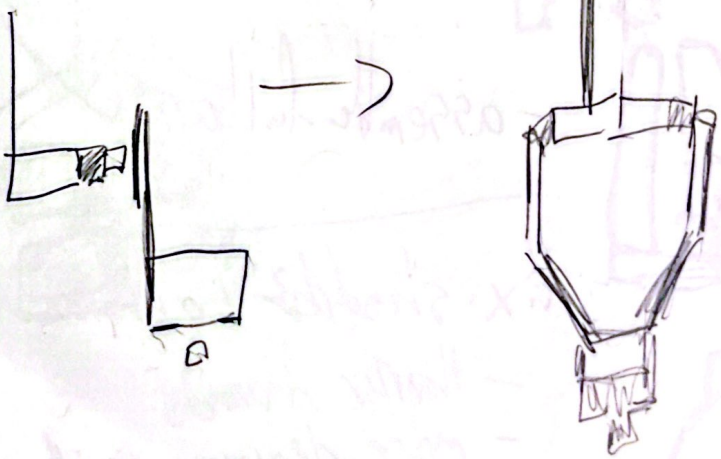
figure out arm wiring

test all motors together

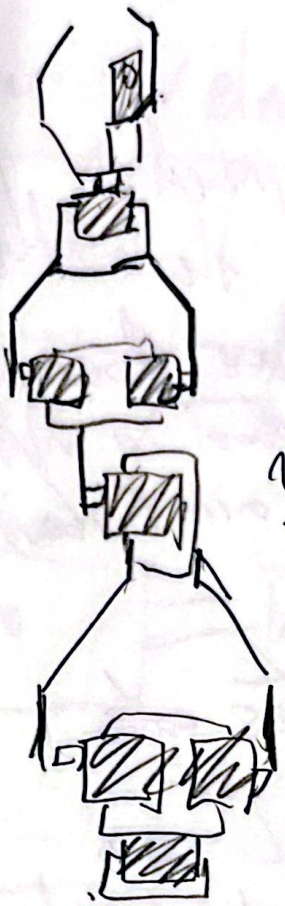
---

elbow to wrist + wrist to hand ✓

redesign



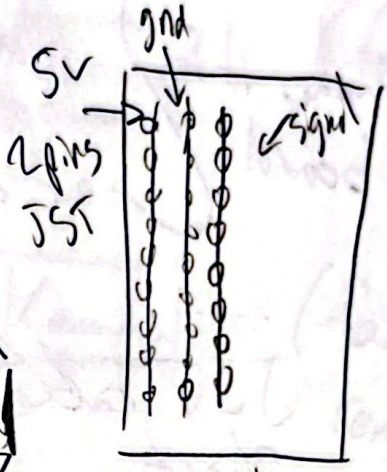
full arm:



Power distribution:

8 motors × 3 wires
(pwr, gnd, signal)

= 24 wires



= 2+8 = 10 pins JST

4 × 4 signal
+ 2 pwr/gnd

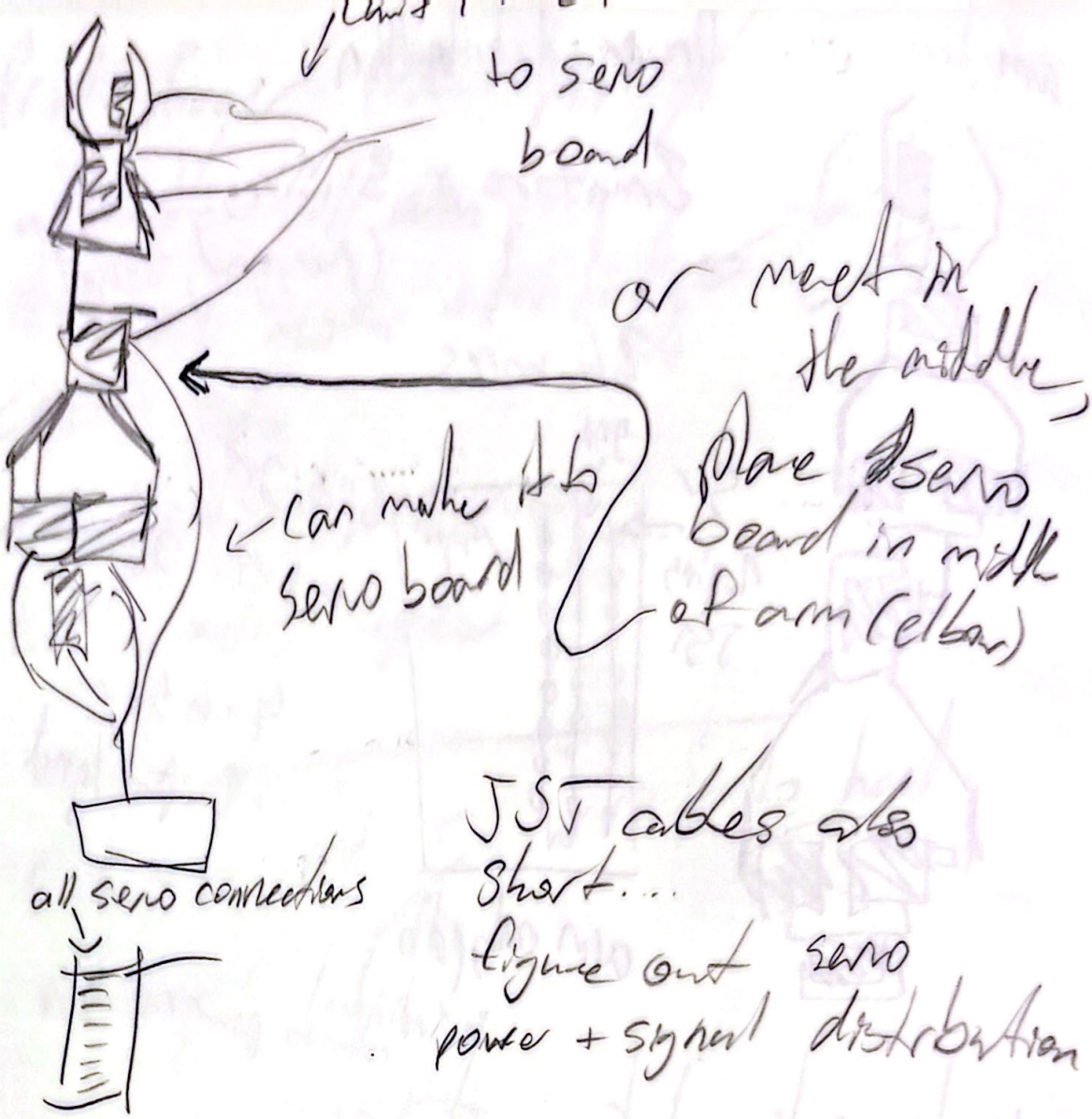
arm only pdb

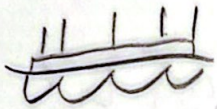
- seperate rails a bit

servos get their own buck converter?
(and motors)

max possible motor draw ~ 10A

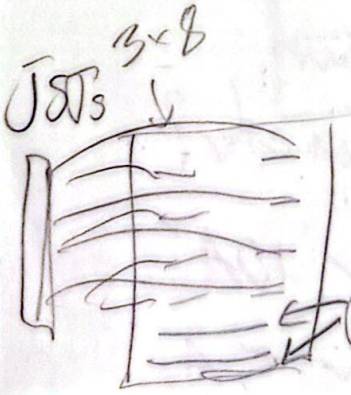
max rasPi + sensor draw ~ 5A



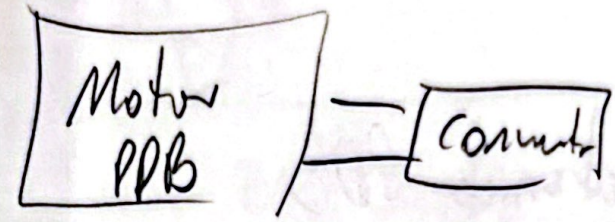


↑ jumper between each pin? tedious but efficient

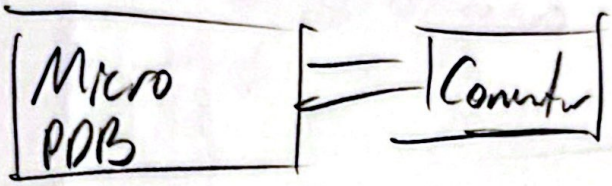
↑ not possible



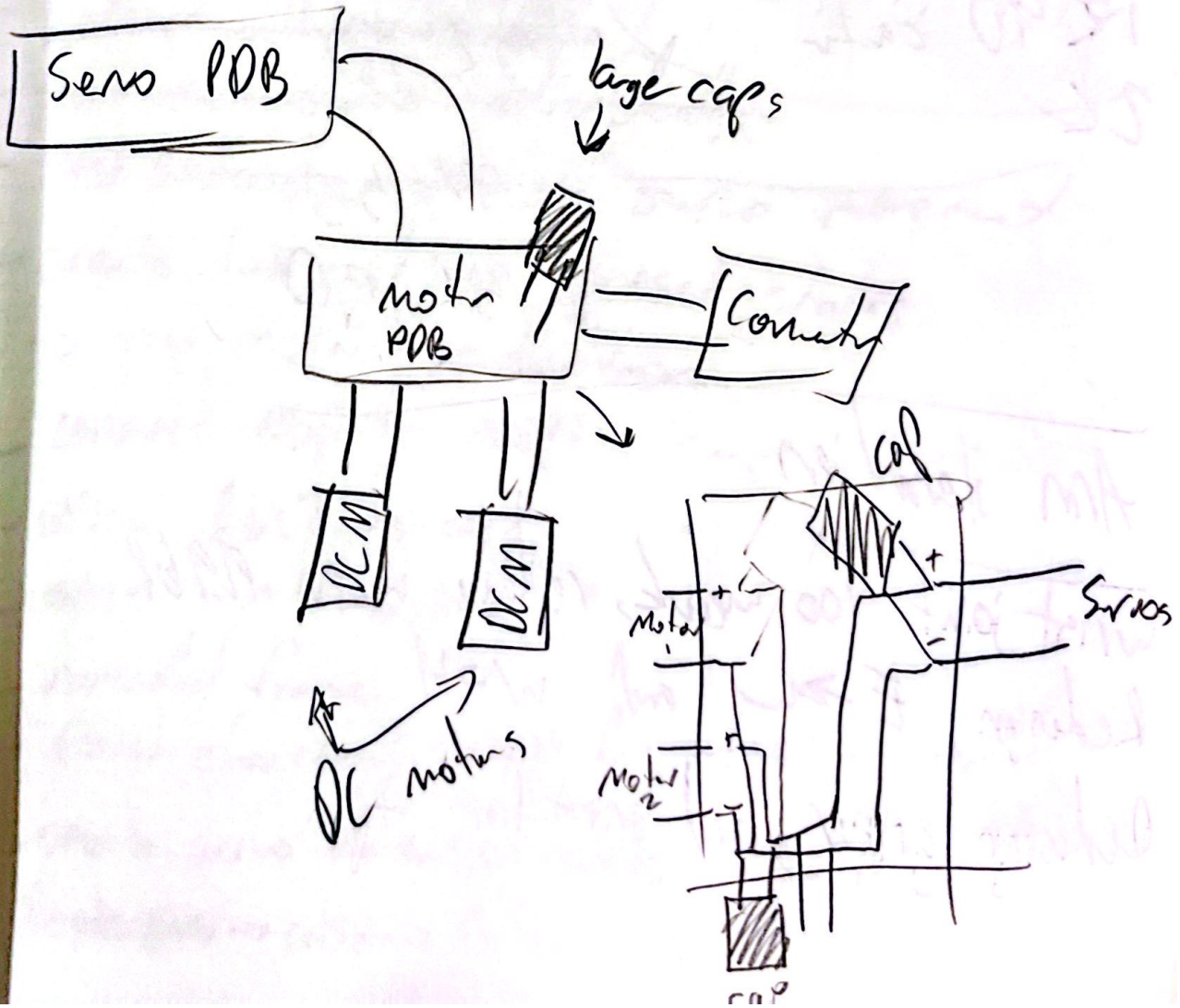
Servo PDB done, need a main PDB for other 5v electronics



(for motors)

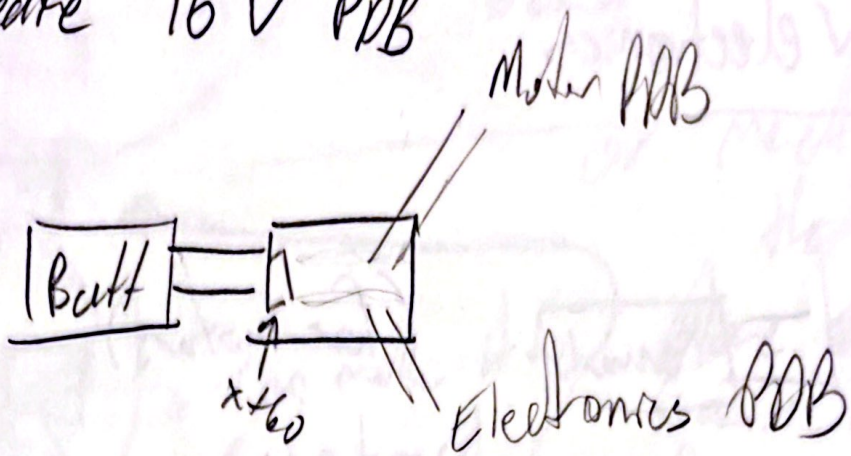


(for rasp + sensors)



Solder JST cables to signal pins

Create 16 V PDB



Create electronics PDB

Motor arm test

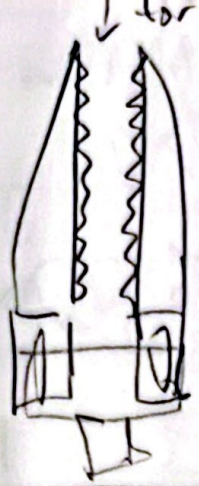
0 < 0 open
1 < 90 center
2 <

0 < 0
3 < 180

0 < 30
3 < 150

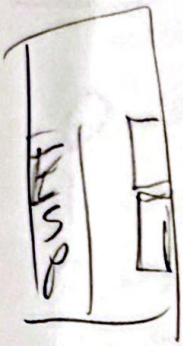
Arm iteration

wrist joint too weak, replace with 996R
Redesign E → W and W → H
Redesign gripper, and print four tips



↓ for better grip

← gripper iteration
2 motors + TPU



ESP mobo

← JST for servo

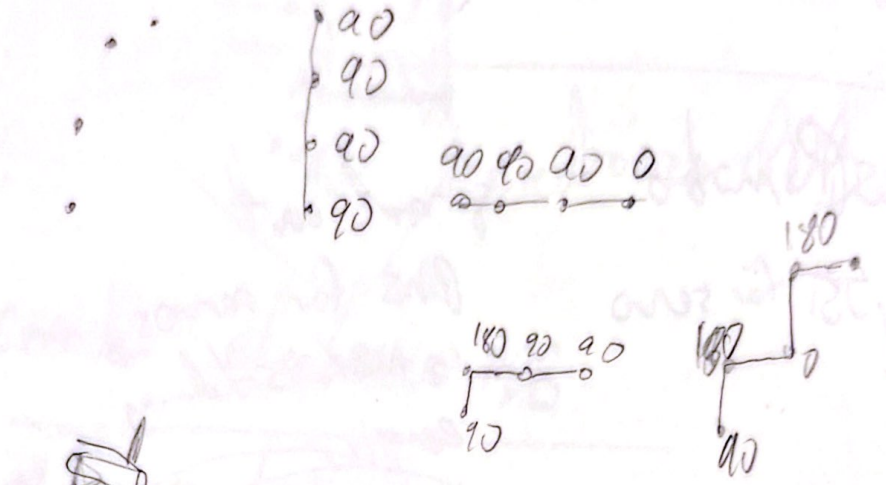
~~spread out~~ pins for servos to make soldering ~~easier~~

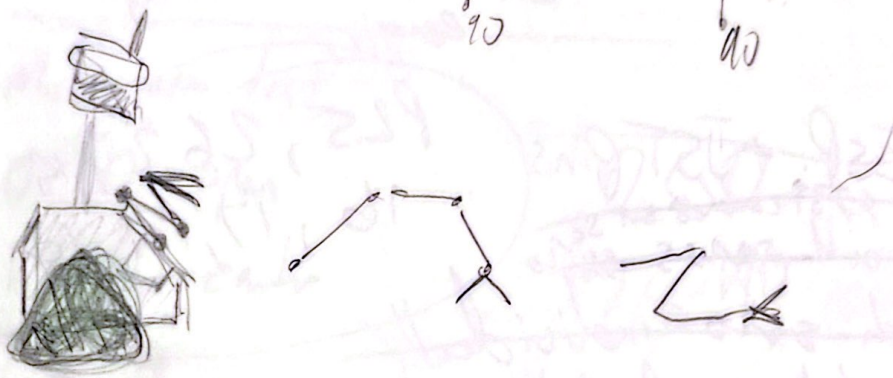
P25, 26, 27, 14
16, 17, 18, 19

- ~~Solder ESP + JST pins~~
- ~~solder test JST cables on servo~~
- ~~Connect all servos etc~~
- ~~test each servo individually~~
- ~~test shoulder double servo movement~~
- create full "actions", preset states to test motor coordination
- connect esp to raspi
- install Ros2 on raspi
- control arm through Ros 2
- remodel frame
- finish electrical things for wheels (encoders, PDB, etc)
- create servo extension cables (male pins → cable → female pins), taking servo pins off frame

16-Q, 17-P, 18-N, 19-O

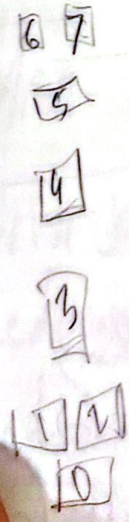
14-Q 27-P 25-N 26-O

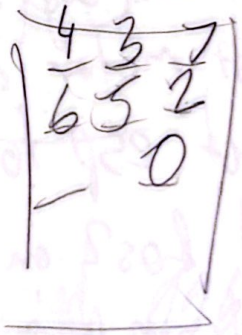


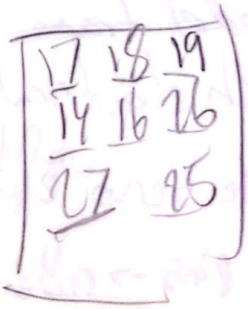


motor mappings -

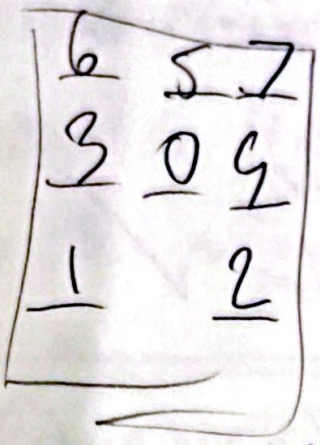
#	Bin #
0	25
1	27
2	26
3	18
4	17
5	16
6	14
7	19







New wiring standard:



kind of resembles arm shape

	Pin#
0 →	16
1 →	27
2 →	25
3 →	14
4 →	26
5 →	18
6 →	17
7 →	19

} arduino pins array

Inverse kinematics

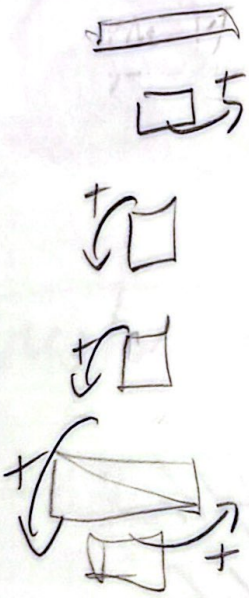
MoveIt2 - looks well supported, nightmare to compile, bulky + hard to use

KDL - shallish, easy to compile, supported decently

OIKFast - can't figure out how to compile

KDL - create chain from URDF,
create solver with max tries
Joint array, run solver
publish output joint states ✓

- match urdf to actual motor direction
- create serial comm node ✓
- test IK !!! ✓



- change 90° → 0°

-90 - 90

not

0 - 180