Robotic assembly using impedance control and in-hand 3D perception

James Watson[1,2], Austin Miller[2] and Nikolaus Correll[1,2]

[1]University of Colorado at Boulder

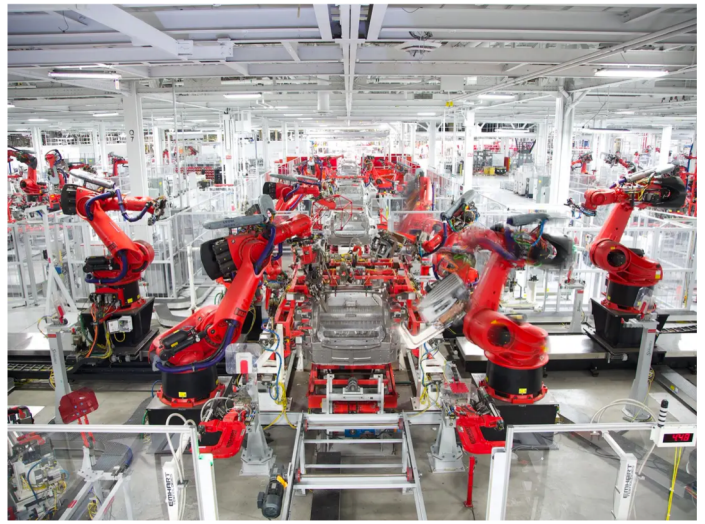
[2]Robotic Materials Inc.

Challenge: if its worth it, built a machine

Manual assembly

Shared autonomy

Fully autonomous



Increasing quantity

As soon as you make enough items, a custom machine maximizes throughput

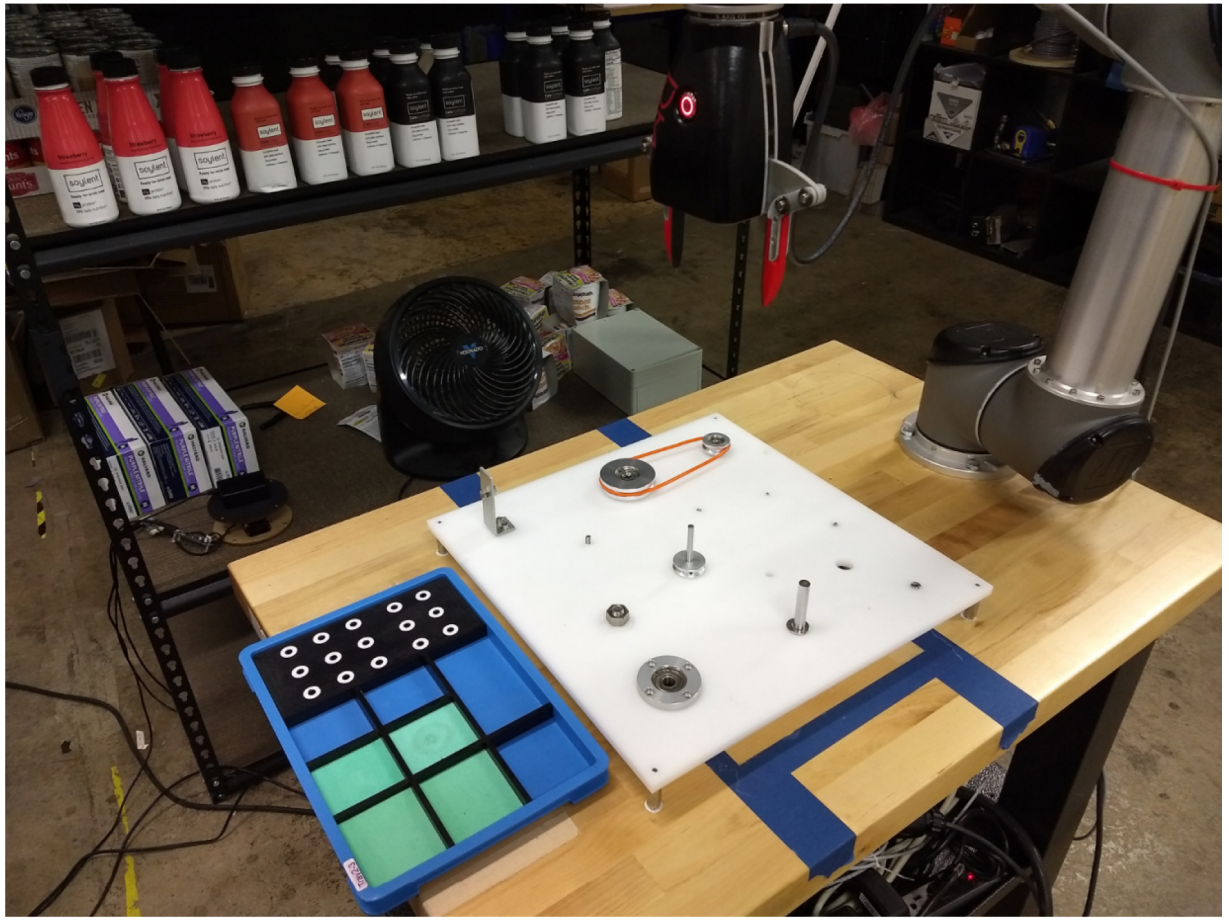
Drivers for “level 5” automation

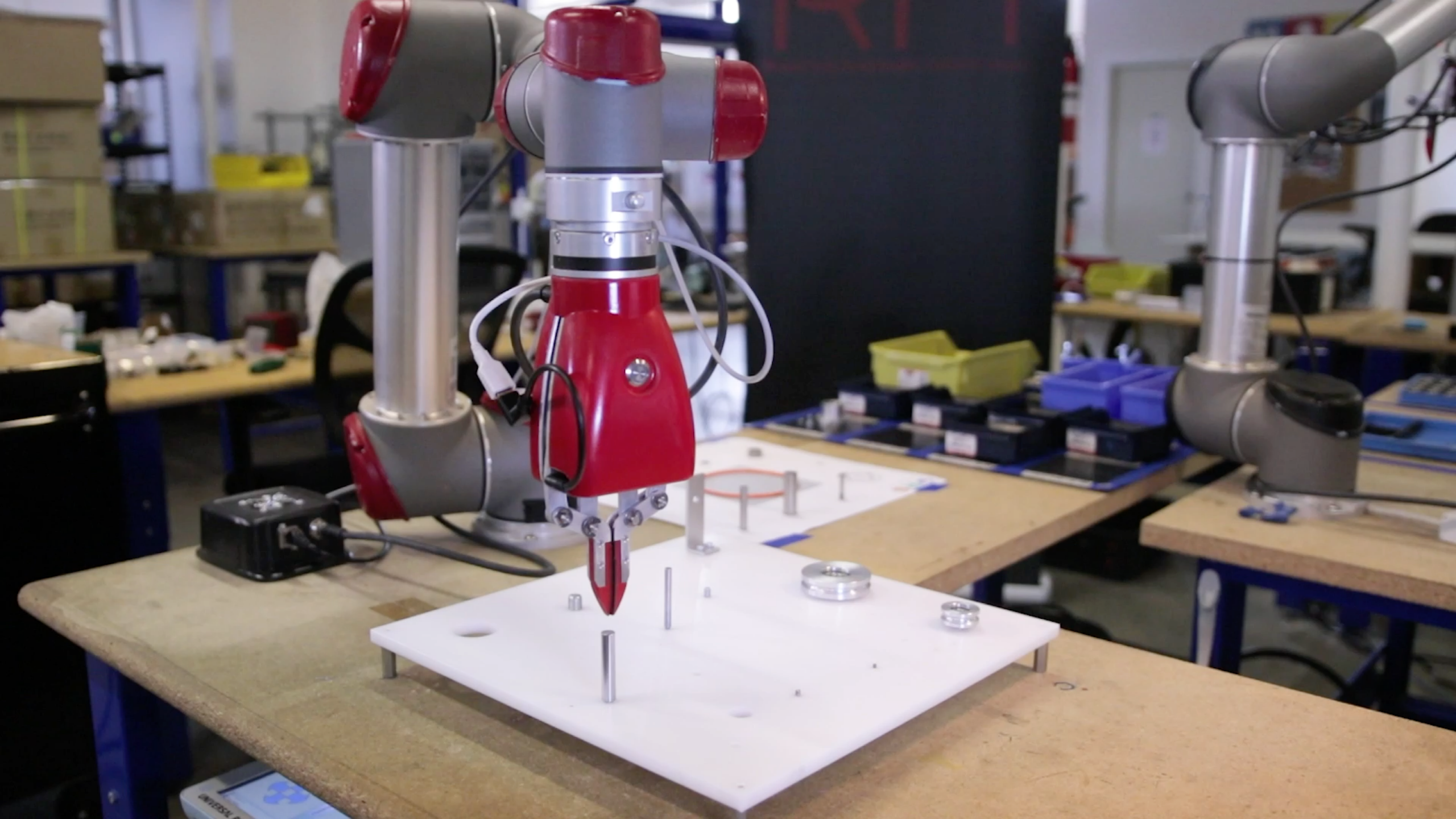
- Most people don’t sell enough to justify building a custom machine -> they need one machine to do multiple things
- Time to setup a production system needs to be minimized
- Tasks with a high degree of automation might only need sporadic tending
- Future: automated maintenance at home, work, and in the field

There is demand for human-like general manipulation.

World Robot Summit Plate Assembly

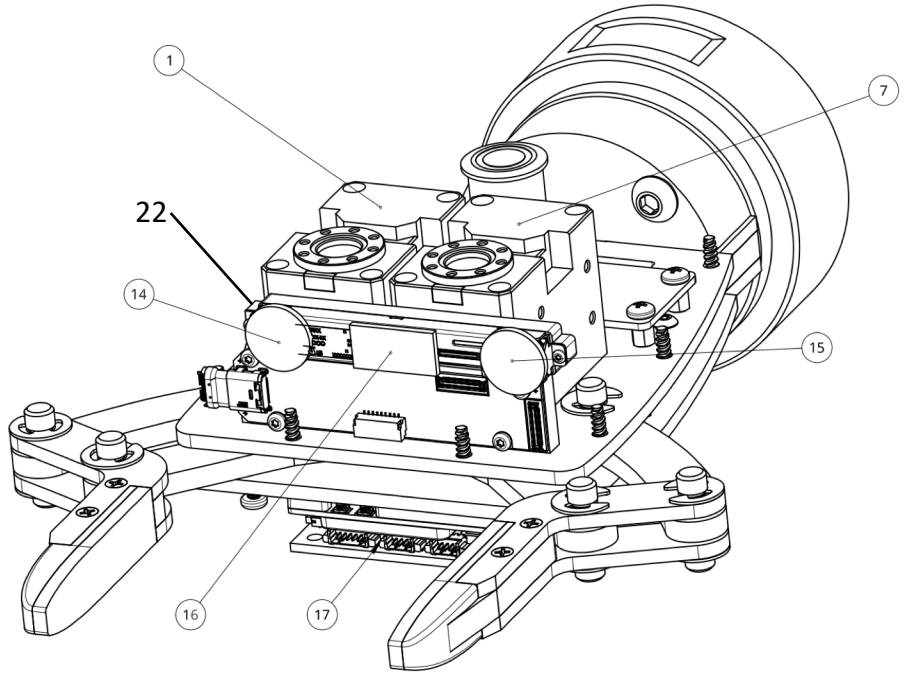
- Assembly of standard machine parts
- Initial pose only roughly known
- Assembly plate can move in-between steps
- At WRS: parts on a kitting mat
- Today: parts presented in ”kitting tray”





Hardware

- UR5 robot with Optoforce load cell
- Robotic Materials SmartHand
  - Intel Realsense
  - Nvidia Jetson
  - Individually impedance-controlled fingers
- Leveraging open-source tools and Python
- Eye-in-hand: constant calibration error
- Stereo depth images as close as 11cm

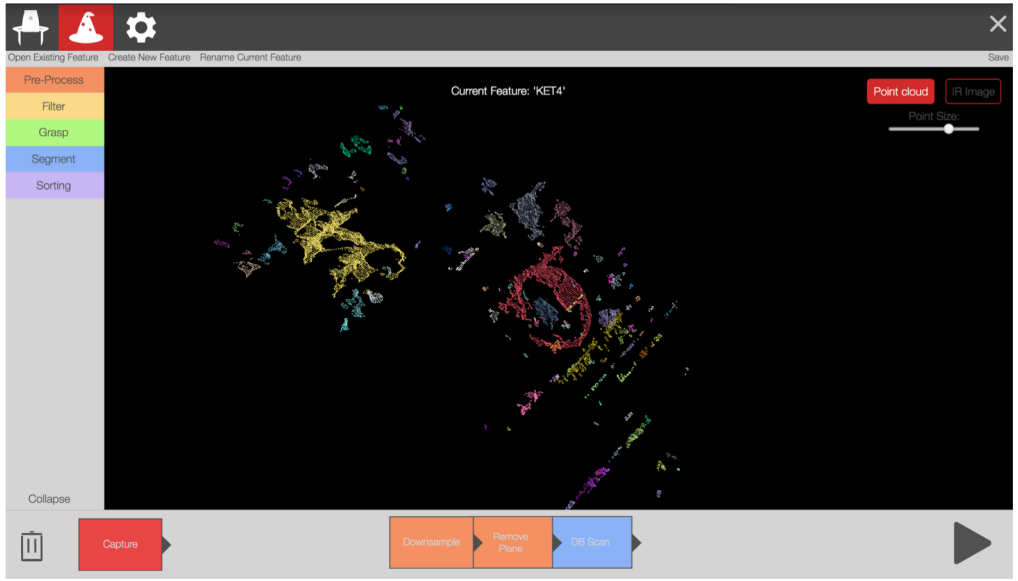


Approach

- Define “features” in the environment using 2D and 3D vision
- Define assemblies in terms of

  feature_1 -> assembly action -> feature_2

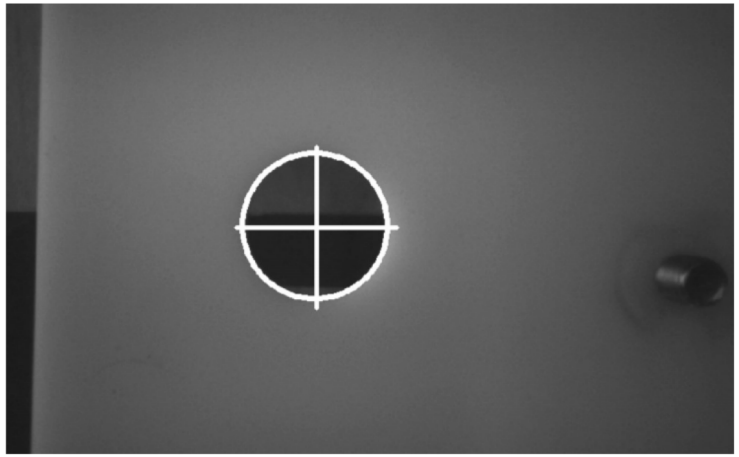
- Use force and torque measurement to define state transitions

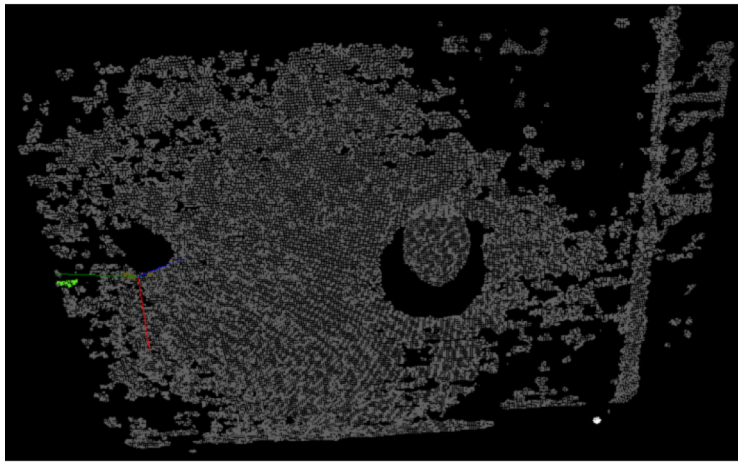


“Feature editor”

Feature detection

- Find circular features (holes) using Hough transform in the image space
- Find pegs using point cloud features



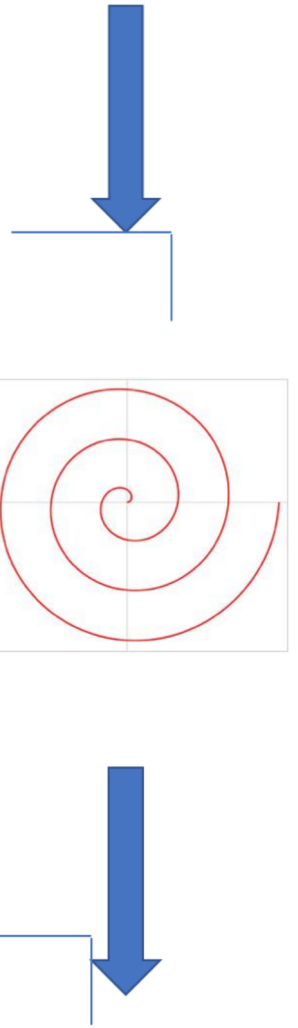


Hole Dia	7cm height	17cm height	27cm height
9mm	0.92	0.99	1.00
17mm	0.59	1.00	1.00
35mm	0.96	1.00	1.00

Spiral-search

Algorithm 1 Algorithm for inserting a part using a spiral movement

1: **procedure** InsertPartSpiral($\vec{x}_i$, Δz, $F_t = 1$, $F_d = 1$, $F_i = 2$, Δ_{max})
2:
3: **function** MaxDownForce(f) ▷ Used as a stopping criterion when moving down
4: **return** $GetWristForce(Z) < f$
5: **end function**
6:
7: **function** MaxSpiralForces ▷ Used as a stopping criterin during spiraling
8: **return** $(\|GetWristTorque(X)\| > 0.9) \vee (\|GetWristTorque(Y)\| > 0.9) \vee (GetWristForce(Z) > -F_d)$
9: **end function**
10:
11: **MoveAbs** $(\vec{x}_i + [0, 0, \Delta z, 0, 0, 0])$
12: **MoveRel** $([0, 0, -\Delta_{max}, 0, 0, 0], v = 0.01) \leftarrow$ MaxDownForce$(-F_t)$
13: **MoveSpiral**$(x_0, y_0, \Delta s = 0.00001, r_{max} = 0.004, v = 0.002, a = 0.5) \leftarrow$ MaxSpiralForces()
14: **MoveRel** $([0, 0, -\Delta_{max}, 0, 0, 0], v = 0.01) \leftarrow$ MaxDownForce$(-F_i)$
15: **end procedure**



Tilt-based assembly

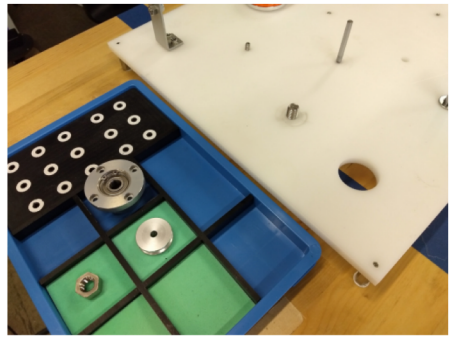
Algorithm 2 Algorithm for inserting a part using a tilting movement

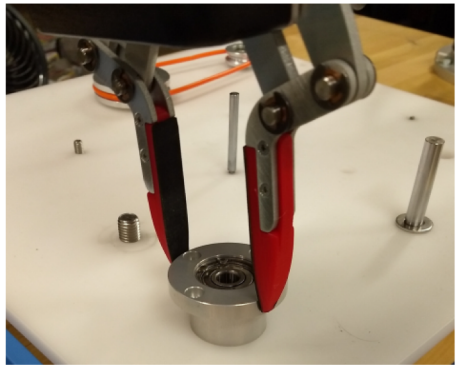
1: **procedure** INSERTTILT($\vec{x}_{hole}$, Δz , θ_{tilt} , Dia , Δx , F_{touch} , F_{insert} , Δ_{max})
2:
3: **MoveAbs** $(\vec{x}_{hole} + [0, 0, \Delta z, 0, 0, 0])$
4: $x_{offset} \leftarrow \Delta x + \sin\theta_{tilt}$
5: **MoveRel**($[-(x_{offset} + \text{Dia}/4), 0, 0, 0, 0, 0]$)
6: **MoveRel**($[0, 0, 0, 0, \theta_{tilt}, 0]$)
7: **MoveRel** $([0, 0, -\Delta_{max}, 0, 0, 0]) \leftarrow$ MAXDOWNFORCE$(-F_{touch})$
8: **MoveRel**($[(x_{offset}+\text{Dia}/4), 0, 0, 0, -\theta_{tilt}, 0]$) $\leftarrow$ MAXDOWNFORCE$(-F_{insert})$
9: **OpenGripper**()
10: **MoveRel** $([0, 0, \Delta_{max}, 0, 0, 0])$
11: **end procedure**

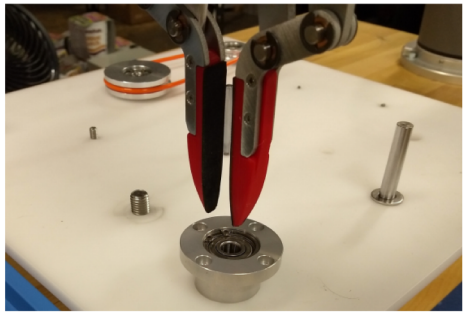
Example: insert bearing algorithm

Algorithm 3 Algorithm for the Bearing Insertion Task

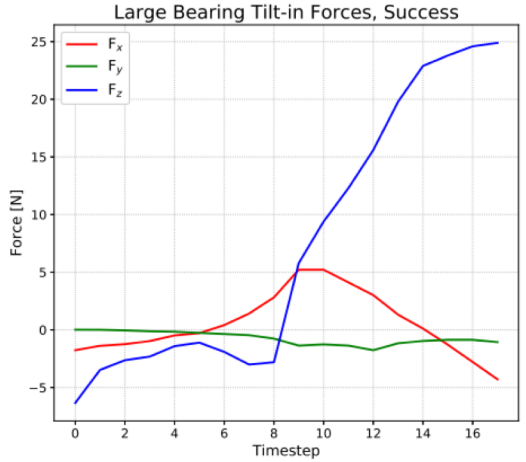
1: **procedure** BEARINGTASK($\vec{x}_{BrgView}$, $\vec{x}_{HolView}$, $\text{Dia}_{Bearing}$, Dia_{Hole} , z_{view} , $z_{PickDepth}$, Δz , θ_{tilt} , F_{touch} , F_{insert} , Δ_{max} , $d_{fingerSep}$, $\vec{x}_{safe}$)
2:
3: **MoveAbs**($\vec{x}_{BrgView}$)
4: $\vec{x}_{BrgLoc} \leftarrow$ CIRCLELOCATOR($\text{DIA}_{Bearing}$, z_{view})
5: **MoveAbs**($\vec{x}_{HolView}$)
6: $\vec{x}_{HolLoc} \leftarrow$ CIRCLELOCATOR(DIA_{Hole} , z_{view})
7: **Pickpart**($\vec{x}_{BrgLoc}$, $\text{Dia}_{Bearing}$, $z_{PickDepth}$)
8: **InsertTilt**($\vec{x}_{HolLoc}$, Δz , θ_{tilt} , Dia_{Hole} , Δx , F_{touch} , F_{insert} , Δ_{max})
9: **OpenGripper**()
10: **SetGripperOpen**($d_{fingerSep}$)
11: **TampWithForceLimit**($\vec{x}_{HolLoc}$, Δz , F_{touch})
12: **MoveAbs**($\vec{x}_{safe}$)
13: **end procedure**

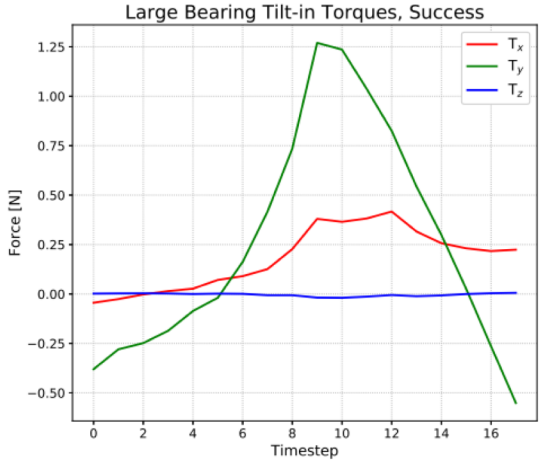


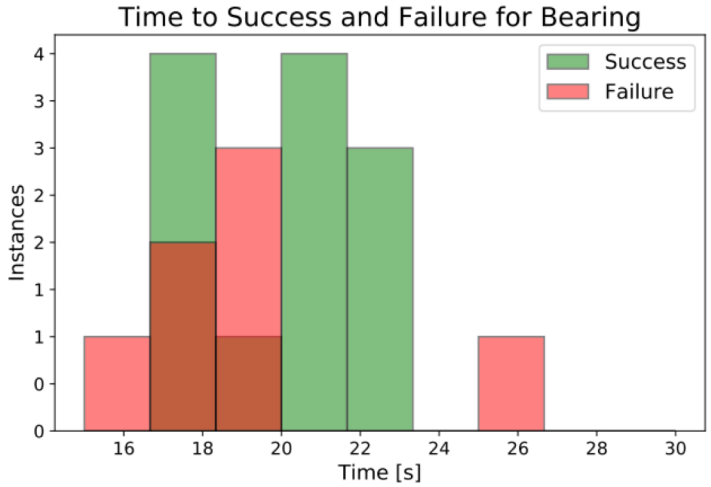


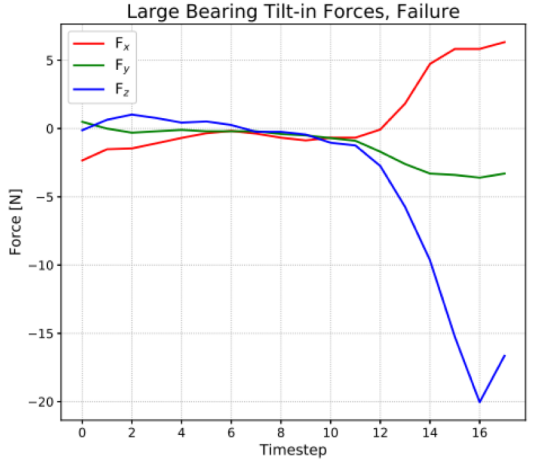


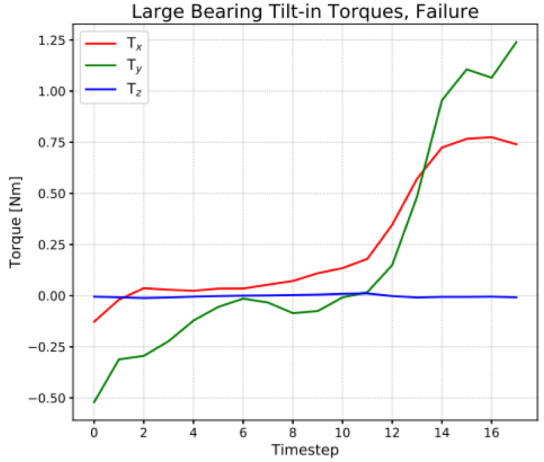
Results: bearing assembly











Task	Trials	Success	Rate
Large Bearing	20	13	0.65
Small Pulley	20	13	0.65
Stud	20	7	0.35

Example: insert small pulley

Algorithm 4 Algorithm for the Small Pulley Task

1: **procedure** PULLEYTASK($\vec{x}_{ShaftView}$, $\vec{x}_{PullyView}$, Dia_{Shaft} , Dia_{Pulley} , $z_{PickDepth}$, Δz , $F_t = 1$, $F_d = 1$, $F_i = 2$, Δ_{max} , $d_{TampOffset}$, F_{touch} , $d_{fingerSep}$, d_{offset} , z_{grasp} , $\vec{x}_{safe}$)

2:

3: **MoveAbs**($\vec{x}_{ShaftView}$)

4: $\vec{x}_{ShftLoc} \leftarrow$ CIRCLELOCATOR(DIA_{Shaft} , z_{view})

5: **MoveAbs**($\vec{x}_{PullyView}$)

6: $\vec{x}_{PulyLoc} \leftarrow$ CIRCLELOCATOR(DIA_{Pulley} , z_{view})

7: **Pickpart**($\vec{x}_{PulyLoc}$, Dia_{Pulley} , $z_{PickDepth}$)

8: **InsertPartSpiral**($\vec{x}_{ShftLoc}$, Δz, $F_t = 1$, $F_d = 1$, $F_i = 2$, Δ_{max})

9: **MoveAbs**($\vec{x}_{safe}$)

10: **SetGripperOpen**($d_{fingerSep}$)

11: **TampWithForceLimit**($\vec{x}_{ShftLoc} + [0, -d_{offset}, 0, 0, 0, 0]$, Δz , F_{touch})

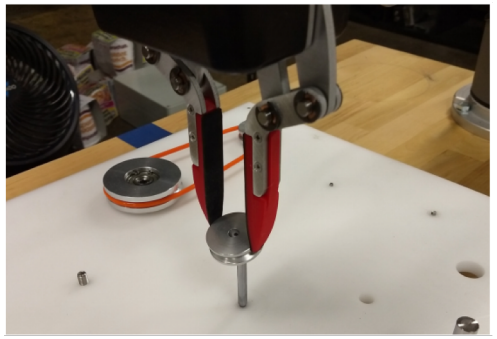
12: **TampWithForceLimit**($\vec{x}_{ShftLoc} + [0, d_{offset}, 0, 0, 0, 0]$, Δz , F_{touch})

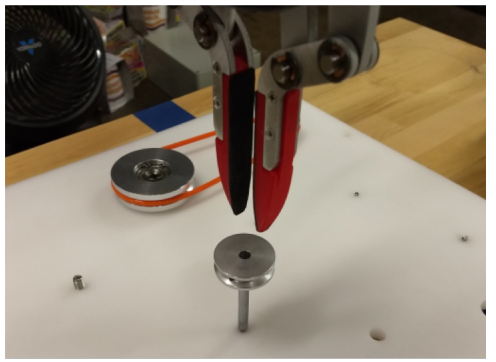
13: **TampWithForceLimit**($\vec{x}_{ShftLoc}$, Δz , F_{touch})

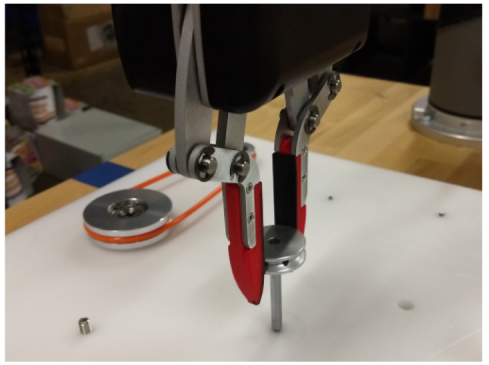
14: **Twist**($\vec{x}_{ShftLoc}$, θ_{osc} , N_{osc})

15: **MoveAbs**($\vec{x}_{safe}$)

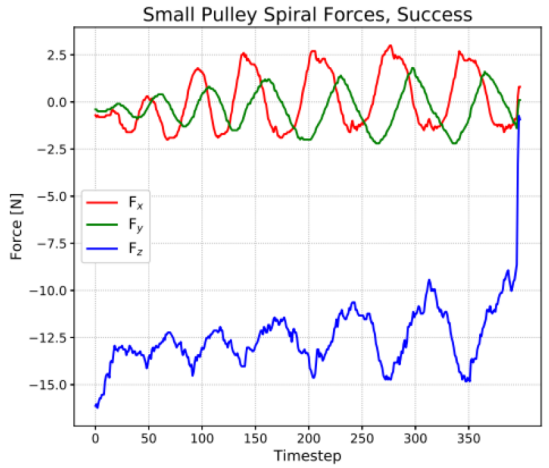
16: **end procedure**

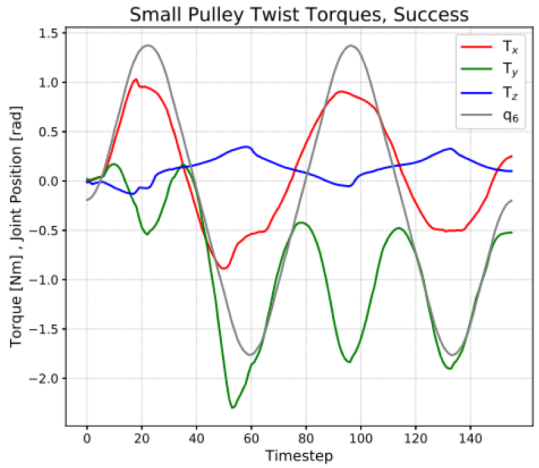


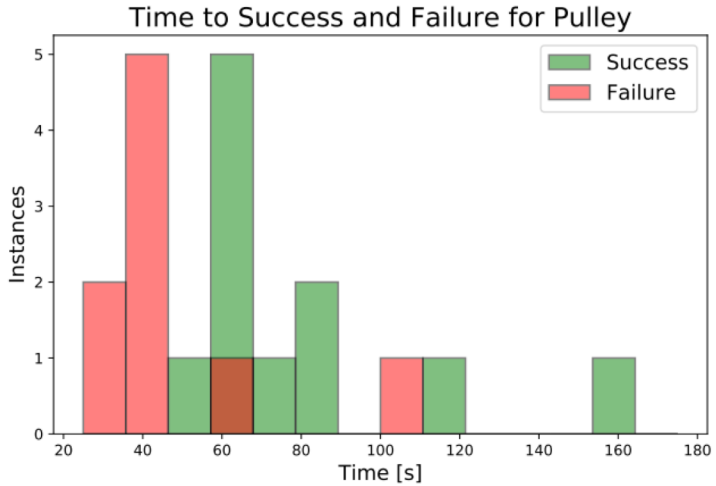


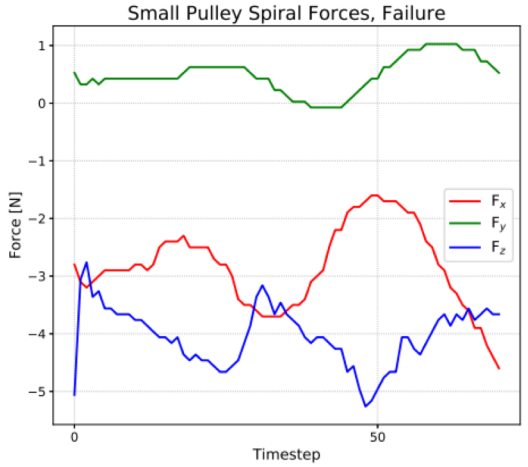


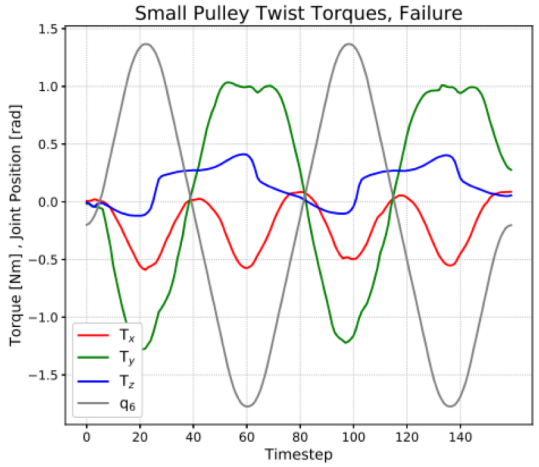
Results: Pulley assembly









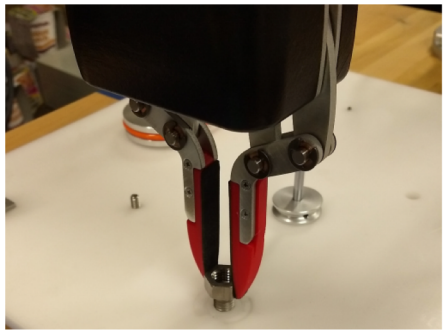


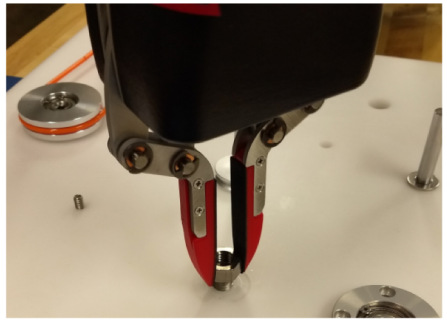
Task	Trials	Success	Rate
Large Bearing	20	13	0.65
Small Pulley	20	13	0.65
Stud	20	7	0.35

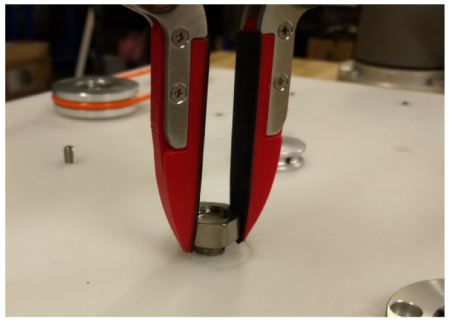
Example: nut threading

Algorithm 5 Algorithm for the Nut Threading Task

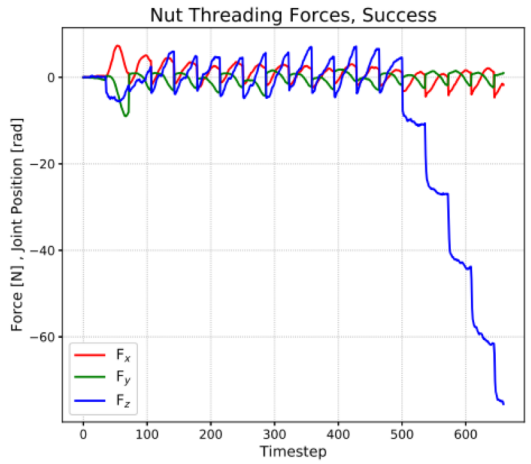
1: **procedure** NUTTHREADING($\vec{x}_{StudView}$, $\vec{x}_{NutView}$, Dia_{Stud} , Dia_{Hole} , Dia_{Nut} , $z_{PickDepth}$, Δz , $F_t = 4$, $F_d = 4$, $F_i = 2$, Δ_{max} , N_{flats} , $T_{z,limit}$, d_{pitch} $\vec{x}_{safe}$)
2:
3: **MoveAbs**($\vec{x}_{StudView}$)
4: $\vec{x}_{StudLoc} \leftarrow$ CIRCLELOCATOR(DIA_{Stud} , z_{view})
5: **MoveAbs**($\vec{x}_{NutView}$)
6: $\vec{x}_{NutLoc} \leftarrow$ CIRCLELOCATOR(DIA_{Hole} , z_{view})
7: **Pickpart**($\vec{x}_{NutLoc}$, Dia_{Nut} , $z_{PickDepth}$)
8: **InsertPartSpiral**($\vec{x}_{StudLoc}$, Δz, $F_t = 4$, $F_d = 4$, $F_i = 2$, Δ_{max})
9: **MoveAbs**($\vec{x}_{safe}$)
10: **OpenGripper**()
11: **Thread**($\vec{x}_{StudLoc}$, N_{flats} , $T_{z,limit}$, d_{pitch})
12: **OpenGripper**()
13: **MoveAbs**($\vec{x}_{safe}$)
14: **end procedure**

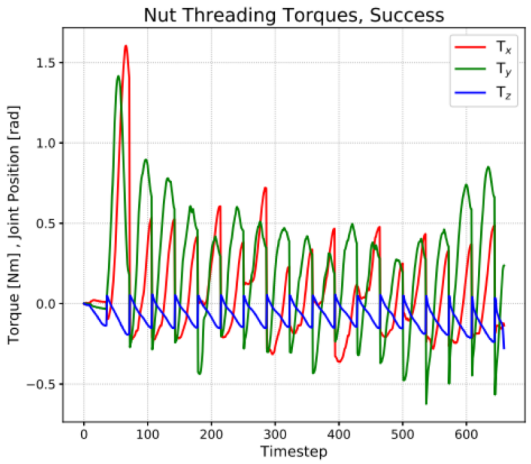


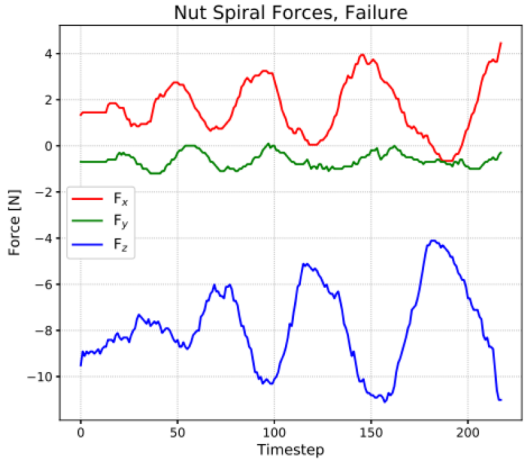


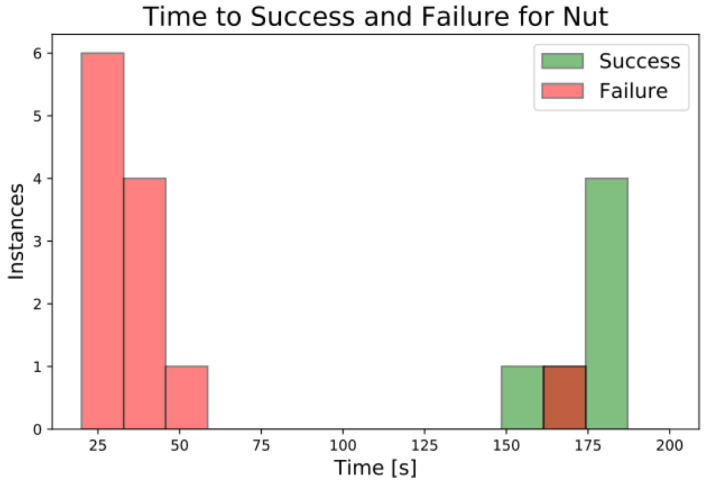


Results: nut threading





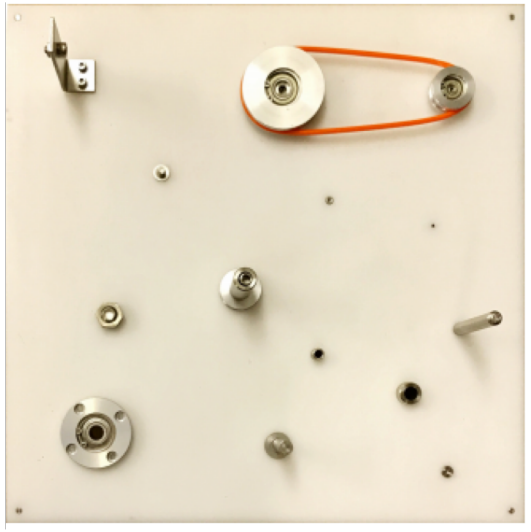




Task	Trials	Success	Rate
Large Bearing	20	13	0.65
Small Pulley	20	13	0.65
Stud	20	7	0.35

Transferring skills

- Two months full-time to move from WRS to NIST challenge
- New tasks
  - Screwing in screws
  - Rectangular parts
  - Cables
  - Gear meshing
- Disassembly
- Is the space of possible parts finite: yes!
- Is the space of possible assemblies finite: no!



Commercial challenges

- NIST task board disassembly still takes 30+ minutes
- Trading efficiency with availability
- Setting up robot still requires extensive programming and experimentation
- Still better to make a custom machine than employing a general robot

Applied research: commercial = research challenges

What are good metrics?

- Reliability (a custom machine is always better)
- Speed (a custom machine is always better)
- **Versatility** – how?
- Size – why?
- Volume – why?
- Price – compound metric reflecting reliability, speed, versatility

Future work

- Machine learning to prevent failures
- 3D pose estimation
- Combine reactive controllers with semantic reasoning
- Associate nouns and verbs with “common sense” knowledge
  - “Screw”: maintain constant force (press down)
  - “Disassemble”: make sure you hold on to part
  - “Assemble”: make sure assembly does not move
  - …
- Domain-specific language

Acknowledgments

SBIR·STTR
America's Seed Fund