Agile Trajectory Generation for Tensile Perching with Aerial Robots

Progress Update

- Demonstrations
  - Meeting with Atar this morning.
  - Have made some changes to simulation based on the drone we're using.
  - Currently retraining this.

From Previously

- Issue around wrapping from different sides.
  - Starting position in state space
  - Previous n states
  - Discussion last time.
    - Take advantage of the symmetry in the environment.
    - More mathematical information on the Reward Function.

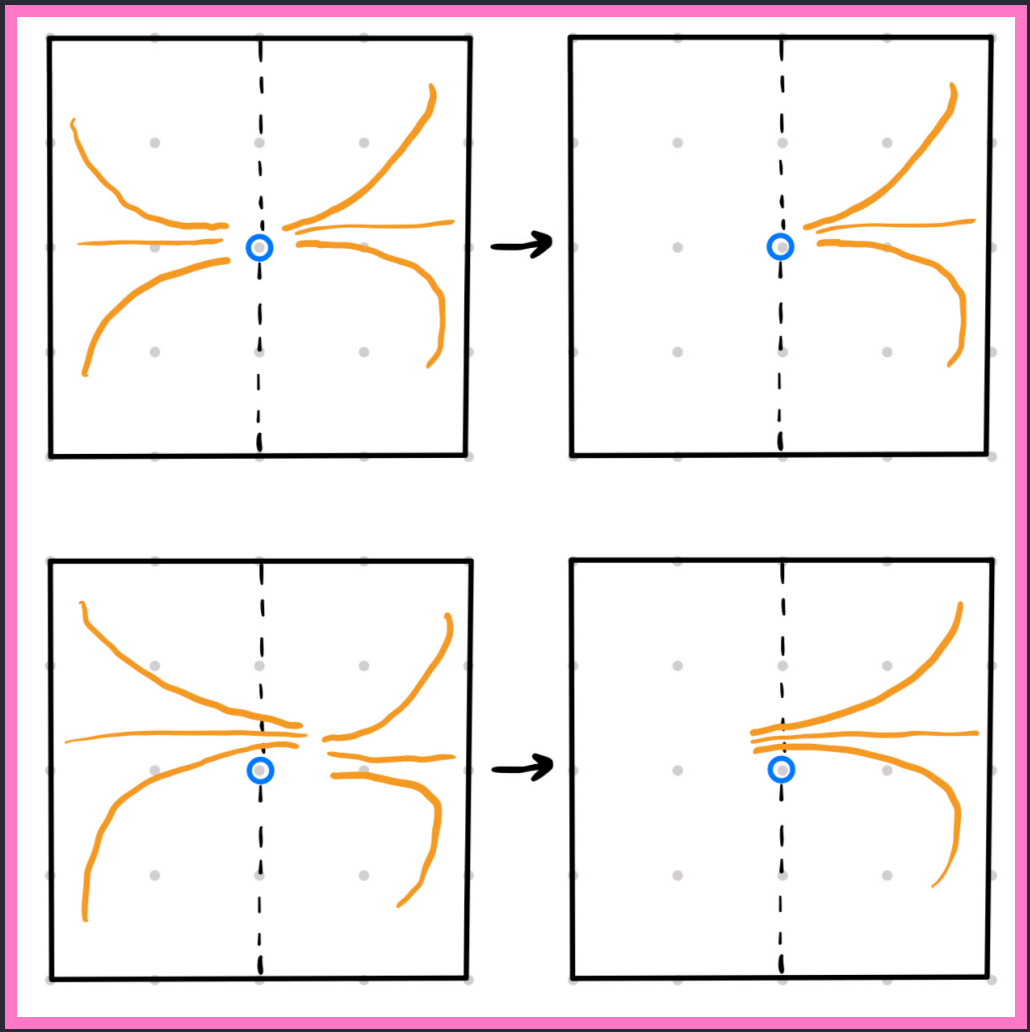
Symmetry

We can assume symmetry across the x=0 plane i.e. from either approaching side of the branch.

Symmetrical Wrapper

- Takes in the positions and actions and converts the positions to be +ve with respect to the starting position.
- Making the problem easier to solve by only needing to learn from one side.
- Implemented as a Gym Wrapper so that it is easy to add/remove to compare the learning effects.

Symmetry Diagram



Reward Function

Reward is currently calculated in levels:

- Approaching: $r_1 = -dist(x_{state}; a_{target})$
- Wrapping: $r_2 = num\_wraps(x)$
- Hanging: $r_3 = max(1 \text{ if } within(x_{state}; h_{box} \text{ else } 0), -dist(x_{state}; h_{target}))$
- Overall:
  - If num_wraps > 1:
    - $r = scale(r_3; -1, 0)$
  - Otherwise:
    - $r = scale(r_2 + r_1; -3, 0)$ otherwise

Further Mathematics

- Distance: $dist(x, target) = norm_{L2}(x, target)$
- Number of Wraps: Algorithm on next slide - based around the position of the two ends of the tether - tracking through different timesteps to calculate rotation.
- Scale: $scale(x; min_x, max_x, a, b) = ((x - min_x)/(max_x - min_x)\ \mathrm{x}\ (b - a))$

Num Wraps

```python
def compute_total_rotation(self):
    pos, _ = p.getBasePositionAndOrientation(self.segments[-1])
    last_x = pos[0]
    last_y = pos[2]
    delta_x = last_x - 0
    delta_y = 2.7 - last_y

    # Compute the angle using arctan2, which considers quadrant location
    angle_radians = np.arctan2(delta_x, delta_y)  # swapped x and y to align with the vertical
    angle_degrees = np.degrees(angle_radians)

    if self.prev_angle is not None:
        # Calculate angle change considering the wrap around at 180/-180
        angle_change = angle_degrees - self.prev_angle
        if angle_change > 180:
            angle_change -= 360
        elif angle_change < -180:
            angle_change += 360

        # Update cumulative angle change
        self.cumulative_angle_change += angle_change

        # Update wraps as a float
        self.wraps = self.cumulative_angle_change / 360.0

    # Update the previous angle for the next call
    self.prev_angle = angle_degrees

    return abs(self.wraps)
```

Stages

Approaching

- Speed - visually much shorter and faster trajectories - need to gather some additional data on this for evaluation purposes.

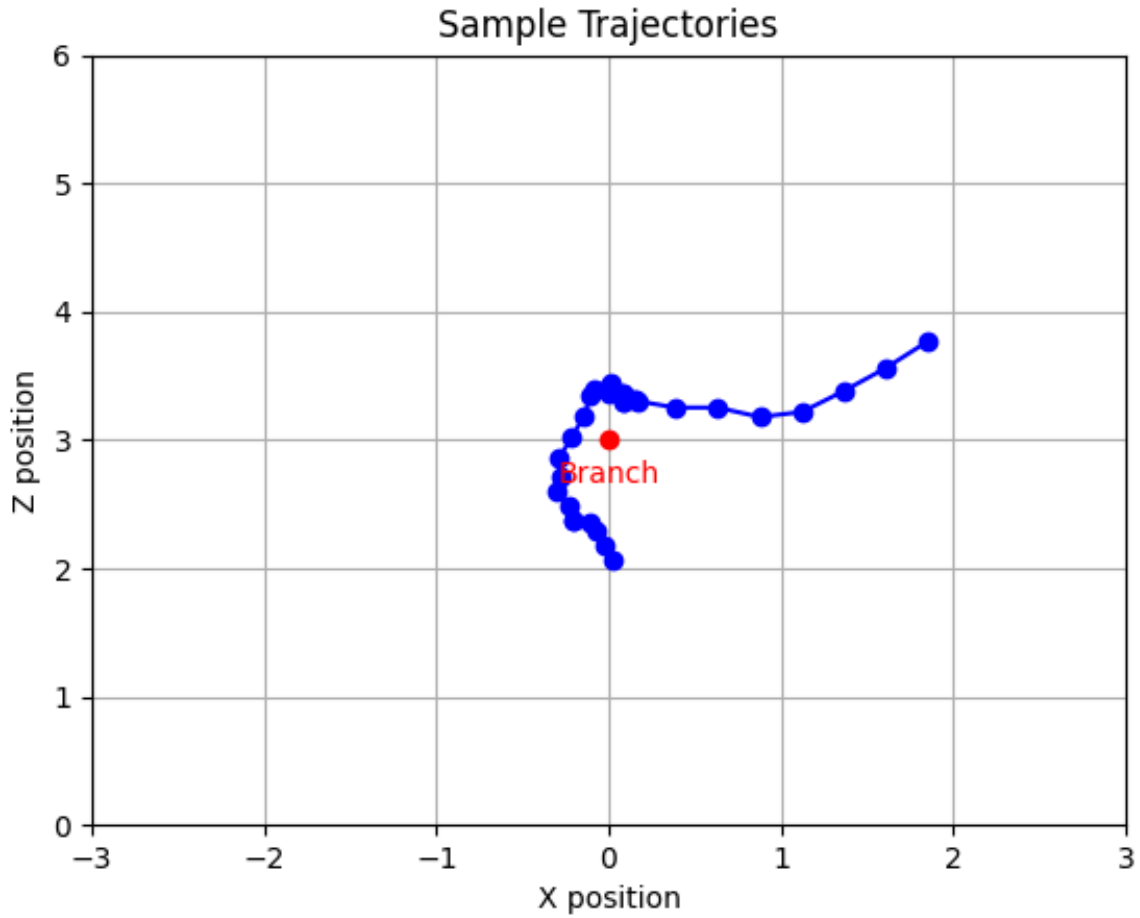
Wrapping

- Waiting
  - Previously using the position of the weight which allowed the network to learn when to move onto the next state.
  - In deployment - This would be complex to actually keep track of in real life - want to aviod this being part of the state space so that the agent can make decisions without this additional knowledge.
  - Using position of weight for training in terms of reward but not using for actual state space - means we won't rely on knowing the position in the actual environment.
  - Incorporate previous state information - "hovering steps" - keep track of how long the agent has hovered - make decisions based on time in a learned manner.

Hanging

- Trajectories with a swinging motion underneath.
  - Questions

Trajectory


Sample Trajectories
Z position
X position
Branch
6
5
4
3
2
1
0
−3
−2
−1
0
1
2
3

Report Plan

- Intro - Background
- Methodology
  - Environmental Modelling
    - Initial Environment
      - Pybullet Environment
      - Tether Modelling
    - Wrappers
      - Dimension
      - Symmetry
      - Memory
      - Timestep
  - Training
    - Reward Function Design
    - Algorithms
    - Demonstrations - Comparison of different training techniques.

- Results
  - Trajectory Experiments
  - Speed
- Conclusion

Overall Plan

- Report Deadline 17th June
  - Week 13th - 20th May --------- Finish Wrapping
  - Week 20th - 27th May --------- Demontration Integration & Experiments - Train the different variations previously discussed to compare the difference in learning - Additional plots showing comparisons etc.
  - Week 27th May - 3rd June ----- Evaluation, Experiments, Report
  - Week 3rd - 10th June ---------- Evaluation, Experiments, Report
  - Week 10th - 17th June -------- Report