Precise Throwing Control

Yuchen Zhang, Yunchao Yao, Samuel Li, Yihan Ruan
{yuchenz7, yunchaoy, swli, yihanr}@andrew.cmu.edu

Introduction

- Control a robotic arm to throw an object with precision to a predetermined location
- Throwing objects precisely is a dynamic human skill that can be useful in a variety of robotic tasks
- We solve a non-linear optimization problem for the arm's trajectory and release point
- We show accurate throws in a bomb dropper and 2-D arm thrower example

Background

- Analytical Models
  - Optimizing control by solving an optimization problem based on approximating dynamics
  - More stability and mathematical guarantees
  - Assumes physical properties
  - Computational expensive
  - Depends on accuracy of model and dynamics
- Learning Models
  - Ignoring low-level dynamics and directly optimize for task-level success signal
  - Can scale to higher dimensional settings
  - More data requirements
  - Physical properties might be difficult to approximate

Toy Problem - Bomb Dropper

- State

$$\mathbf{x} = [x \ \dot{x} \ y \ \dot{y}]^T$$

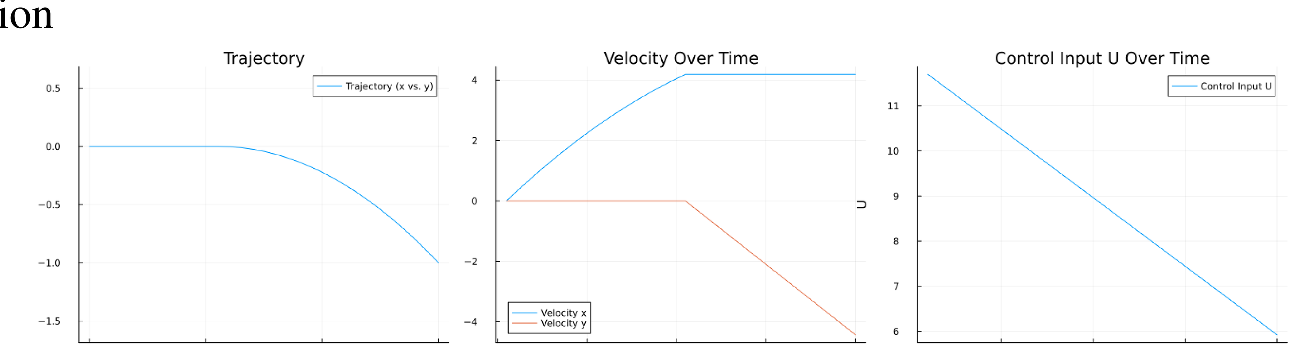
- Dynamic

$$\dot{\mathbf{x}} = f(\mathbf{x}, u) = \begin{cases} \begin{bmatrix} \dot{x} & \frac{u}{m} & \dot{y} & 0 \end{bmatrix} & t < T_1 \\ \begin{bmatrix} \dot{x} & 0 & \dot{y} & -g \end{bmatrix} & t \geq T_1 \end{cases}$$

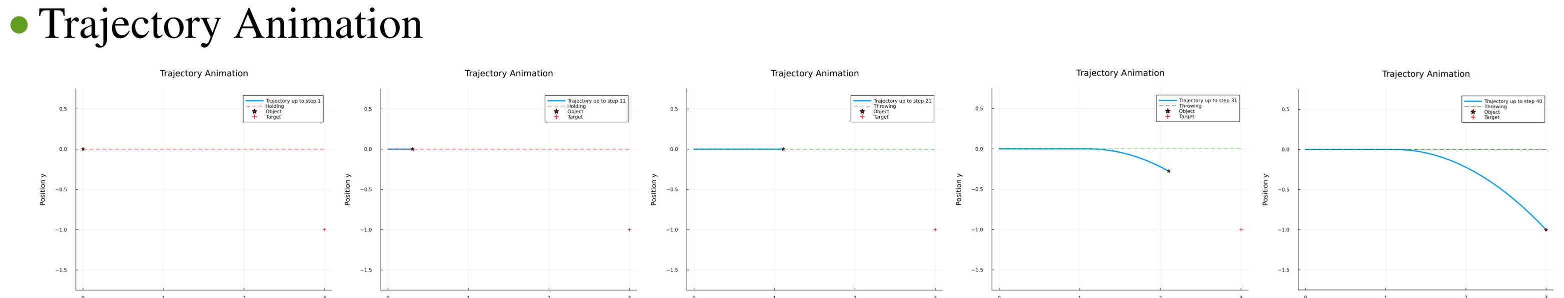
- Non-linear Optimization Problem

$$\begin{aligned} \min_{x_{1:N_1+N_2}, u_{1:N_1}, \Delta t_1, \Delta t_2} \quad & J(x_{1:N_1+N_2}, u_{1:N_1}) = \frac{1}{2}\sum_{i=1}^{N_1} u_i^T R u_i * \Delta t_1 \\ st \quad & x_1 = x_{ic} \\ & x_{N_1+N_2}[1] = goal_x \\ & x_{N_1+N_2}[3] = goal_y \\ & x_{k+1} = f_1(x_k, u_k, \Delta t_1) \quad \text{for } k = 1, \cdots, N_1 \\ & x_{k+1} = f_2(x_k, u_k, \Delta t_2) \quad \text{for } k = N_1+1, \cdots N_1+N_2 \\ & \Delta t_1 \in [0.01, 0.5] \\ & \Delta t_2 \in [0.01, 0.5] \end{aligned}$$

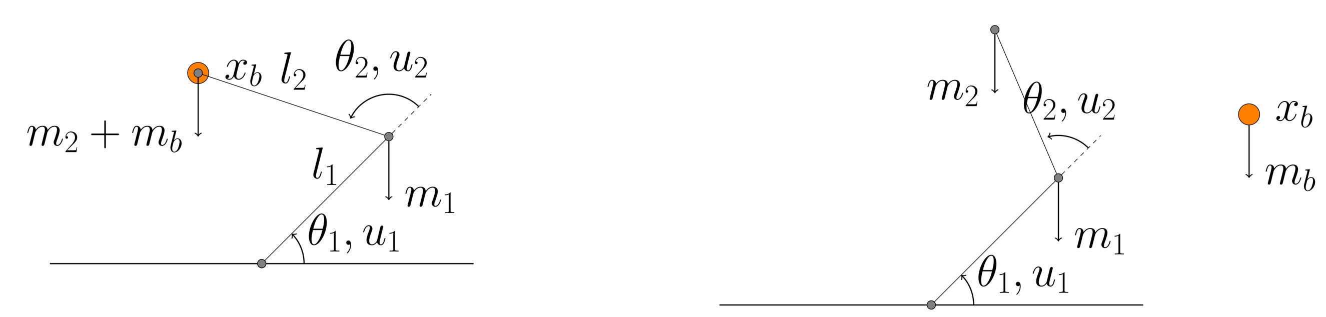
- Solution



- Trajectory Animation

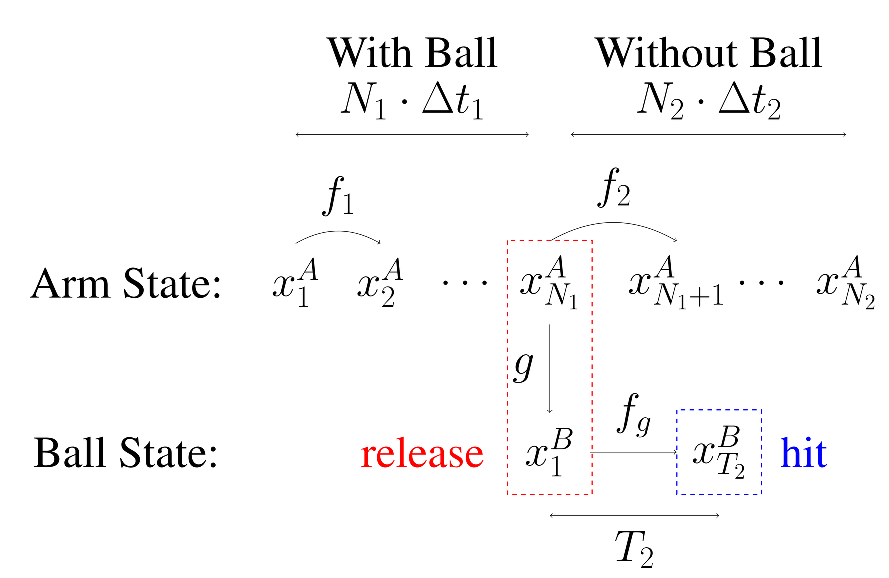


2-D Arm Thrower



- **Optimization Problem Formulation**

We modified the architecture of the bomb dropper for the arm thrower to reduce unnecessary computation for the solver. The discrete states of the optimization problem are as follows:



In the above diagram, x^A represent the arm's state and x^B represent the ball's state. The arm will carry the ball for the first N_1 steps of Δt_1, where it will release the ball to create ball state at release x_1^B. The arm will continue to slow down for N_2 steps of Δt_2, and the ball will travel one step of duration T_2 until it hits the target.

When the arm is carrying the ball, the ball's state is determined by the arm state. We model this dependency as g that maps arm state to ball state. The ball will influence the arm's dynamics with its mass, and we model the dynamics with/without the ball as f_1, f_2. f_g represents ball's freefall dynamics.

We define the arm state as joint angles and velocities, $x^A = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]$, and ball state(when its flying) as XY position and velocity, $x^B = [p_x, p_y, \dot{p}_x, \dot{p}_y]$. The control input u is the torque at joint $1, 2$. We form the optimization problem as:

$$\begin{aligned} \min_{x^A_{1:N_1+N_2}, x^B_{1,T_2}, u_{1:(N_1+N_2-1)}, \Delta t_1, T_2} \quad & J\left(x^A, x^B, u\right) = \frac{1}{2}\sum_{i=1}^{N_1} u_i^T R u_i * \Delta t_1 + \frac{1}{2}\sum_{i=N_1+1}^{N_2-1} u_i^T R u_i * \Delta t_2 \\ st \quad & x_1^A = x_{ic} \\ & x_{k+1}^A = f_1\left(x_k^A, u_k, \Delta t_1\right) \quad \text{for } k = 1, \cdots, N_1 \\ & x_{k+1}^A = f_2\left(x_k^A, u_k, \Delta t_2\right) \quad \text{for } k = N_1+1, \cdots N_1+N_2 \\ & x_1^B = g(x_{N_1}^A) \\ & x_{T_2}^B = f_g(x_1^B, T_2) \\ & x_{T_2}^B[1] = goal_x \\ & x_{T_2}^B[2] = goal_y \\ & \Delta t_1 \in [0.01, 0.5] \\ & T_2 \in [0.1, 10.0] \end{aligned}$$

Where,

$$f_{1,2} = rk4(\tilde{f}_{1,2}, \Delta t_{1,2}), \quad \tilde{f}_{1,2} = M_{1,2}(\theta)^{-1}(u - C_{1,2}(\theta, \dot{\theta}) - G_{1,2}(\theta))$$

represent the arm dynamics with/without the ball. The functions M, C, G of the arm with mass m_1, m_2 can be calculated as:

$$M(\theta) = \begin{bmatrix} (m_1+m_2)l_1^2 + m_2 l_2^2 + 2m_2 l_1 l_2 \cos(\theta_2) & m_2 l_2^2 + m_2 l_1 l_2 \cos(\theta_2) \\ m_2 l_2^2 + 2m l_1 l_2 \cos(\theta_2) & m_2 l_2^2 \end{bmatrix}$$

$$C(\theta, \dot{\theta}) = \left(\begin{bmatrix} 0 & -2m_2 l_1 l_2 \sin(\theta_2) & -m_2 l_1 l_2 \sin(\theta_2) \\ m_2 l_1 l_2 \sin(\theta_2) & 0 & 0 \end{bmatrix}\right) \begin{bmatrix} \dot{\theta}_1^2 \\ \dot{\theta}_1 \dot{\theta}_2 \\ \dot{\theta}_2^2 \end{bmatrix}$$

$$G(\theta) = -g \begin{bmatrix} (m_1+m_2) l_1 \cos(\theta_1) + m_2 l_2 \cos(\theta_1 + \theta_2) \\ m_2 l_2 \cos(\theta_1 + \theta_2) \end{bmatrix}$$

When the arm is not carrying the ball, we can use m_1, m_2 as above to calculate the dynamics. When the arm is carrying the ball, we simply update

$$m_2 \leftarrow m_2 + m_b$$

For the arm-ball state constraint, we can differentiate the kinematics of the arm and get:

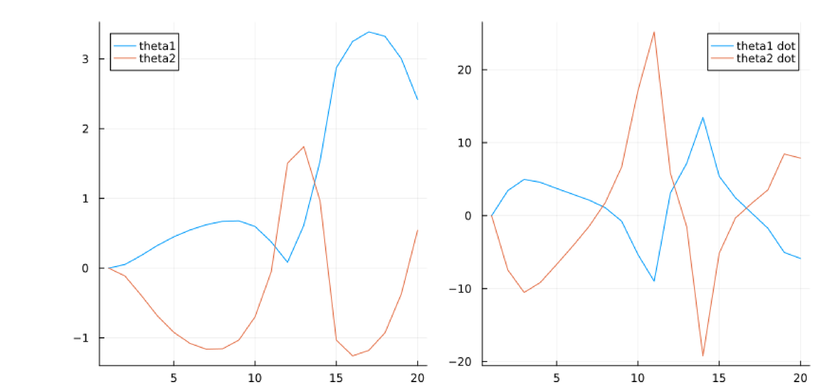
$$\begin{bmatrix} p_x \\ p_y \\ \dot{p}_x \\ \dot{p}_y \end{bmatrix} = \begin{bmatrix} l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \\ l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \\ -l_1 \sin(\theta_1)\dot{\theta}_1 - l_2 \sin(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2) \\ l_1 \cos(\theta_1)\dot{\theta}_1 + l_2 \cos(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2) \end{bmatrix}$$

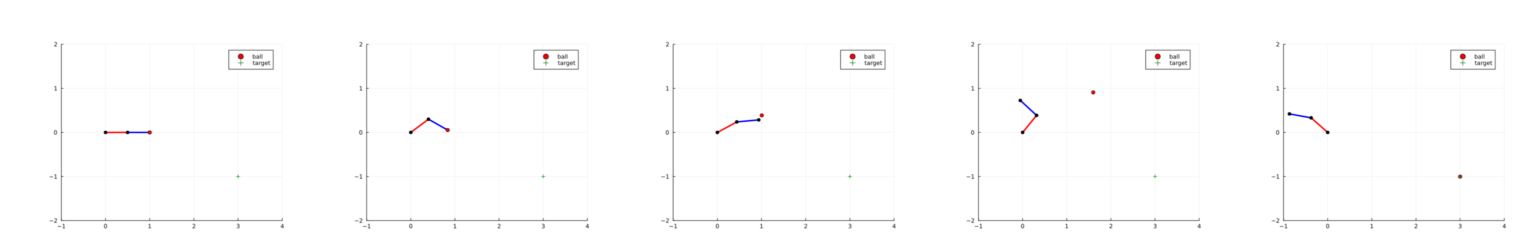
Since the free-fall dynamics is linear, we can integrate it perfectly:

$$f_g(x_1^B, T_2) = [p_x + T_2 \dot{p}_x, \quad p_y + T_2 \dot{p}_y - \frac{g}{2} T_2^2, \quad \dot{p}_x, \quad \dot{p}_y - T_2 g]$$

We solved the problem with Ipopt with $N_1 = N_2 = 10$ steps and $R = 0.01 I_2$.

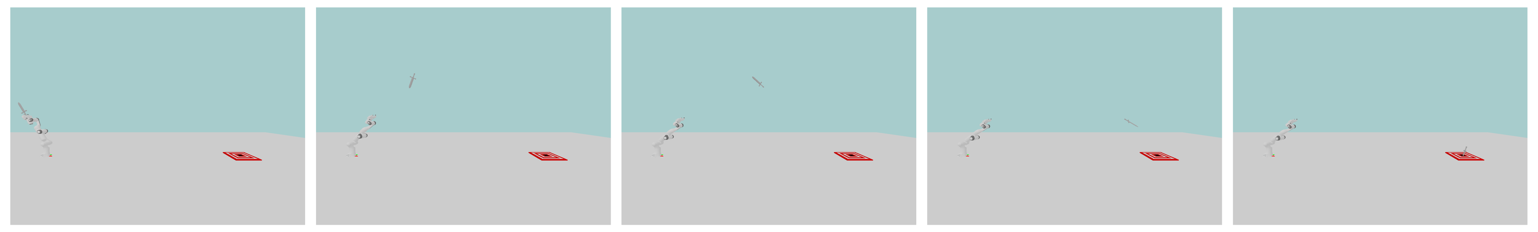
- **Solution and Animation**





Conclusion and Next Steps

- We show how we solve the non-linear optimization problem of throwing an object to hit a target in a 1D bomb drop and 2D arm thrower problem with IPOPT
- However, the performance/result is highly dependent on the initial condition. Therefore, we are planning to provide a valid reference trajectory for stability in convergence
- Next steps for the final report include engineering this approach into a more complex 7DOF arm simulator
- Potentially expand the goal pose to consider orientation in addition to location
- Consider more complex throwing objects, such as an axe or knife



- Maybe applying Iterative Learning Control (ILC) to solve the mismatch problem when changing the simulation environment