Towards Dynamic Robot Juggling: Adaptive Planning and Siteswap Patterns

Dynamisches Roboter Jonglieren: Adaptive Planung und Sitewap Muster
Bachelor thesis by Mario Alejandro Gómez Andreu
Date of submission: July 13, 2023

1. Review: M.Sc. Kai Ploeger
2. Review: Prof. Jan Peters, Ph.D.
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Abstract

Research about athletic movements like juggling has not only been insightful for our understanding of human motor skills but also for improving dynamic dexterity in robots. Robotic juggling has been studied extensively, but successful approaches have been limited to simple juggling patterns indicating missing insights into the additional requirements of more complex patterns.

We present a comprehensive analysis of juggling a wide range of complex juggling patterns - siteswaps - on our two-arm robot. We propose an algorithm that plans hand movement given a desired juggling pattern and is adaptive to ball disturbances due to air movement. We identify two previously unreported constraints to enable siteswap juggling with accurate contact dynamics and open hands (unactuated funnels).

Our system can juggle various siteswap patterns and adapts to non-stationary wind and wind speeds of up to $15\,\mathrm{m\,s^{-1}}$.

Zusammenfassung

Forschung zu athletischer Bewegung hat nicht nur unsere Verständnis von menschlichen Fähigkeiten verbessert, sondern hat es uns auch erlaubt, Roboter mit dynamischem Geschick auszustatten. Roboterjonglieren wurde intensiv untersucht, aber erfolgreiche Ansätze waren bisher auf einfache Jongliermuster beschränkt. Das weißt auf ein fehlendes Verständnis für die zusätzlichen Anforderungen von komplexeren Jongliermustern hin.

In dieser Arbeit präsentieren wir eine umfassende Analyse des Jonglierens einer großen Menge komplexer Jongliermuster so genannte Siteswaps auf unserem zweiarmingen Roboter. Wir stellen einen Algorithmus vor, der für ein gewünschtes Jongliermuster die Armbewegungen plant und sich während des Jonglierens an unerwartete durch Wind verursachte Ballbewegungen anpasst. Wir identifizieren zwei vorher in der Literatur unbekannte Nebenbedingungen, die Siteswap-Jonglieren mit exakter Kontaktdynamik und unbeweglichen trichterförmigen Endeffektoren ermöglicht.

Unser System kann verschiedene Siteswapmuster jonglieren und sich an nicht-stationären Wind und Windgeschwindigkeiten von bis zu $15\,\mathrm{m\,s^{-1}}$ anpassen.

Contents

Glossary

Term	Symbol	Description
ball number	N_{balls}	number of balls that are juggled
ball position	$\boldsymbol{b}$	
cycle duration	T_{cycle}	time between two catches or two throws of one hand
dwell duration	T_{dwell}	time a ball is held in a hand
end effector position	$\boldsymbol{x}$	
excited state		state in the siteswap graph that is not the ground state
flight duration	T_{flight}	time a ball spends in the air
ground state		state in the siteswap graph corresponding to a uniform N_{balls} pattern
hand normal	$\boldsymbol{e}_h$	z-axis of the hand coordinate system
hand number	N_{hands}	number of hands used to juggle
vacant duration	T_{vacant}	time a hand does not hold a ball and is therefore empty
wind speed	$\boldsymbol{w}$	

1. Introduction

For decades, juggling has been a benchmark for the speed and precision of robotic systems. This work presents a dynamic online planner capable of juggling complex siteswap patterns with up to 5 balls in a simulated environment. It can control the balls on stable trajectories even in the presence of non-stationary wind or under challenging simulated contact dynamics.

1.1. Problem Statement

Most patterns we investigate will obey two of the following three assumptions Polster [16] proposes to define *simple juggling*:

(J1) The balls are juggled to a constant beat; that is, the throws occur at discrete, equally spaced moments in time.

(J2) Patterns are periodic, and we can and will assume that our juggler has been juggling forever and will never stop juggling, repeating the same pattern over and over.

(J3) At most one ball gets caught and thrown on every beat and if one is caught, the same ball is thrown.

J1 allows us to formalize juggling as a time-discrete problem, allowing representation in simple transition systems. **J3** excludes multiple balls in one hand. This removes the requirement for in-hand manipulation of individual balls, which the given unactuated end effector is not designed for. **J2** will be relaxed in this work because Polster only considers the repeating pattern (steady state) of a juggling sequence. Our work also includes the transient problem of reaching such a steady state from a standstill. We will therefore adhere to a relaxed version of Polster's definition, which we call *semi-simple juggling*:

(J1) The balls are juggled to a constant beat; that is, the throws occur at discrete equally spaced moments in time.

(J2') Patterns are periodic once they reach their steady state.

(J3) At most one ball gets caught and thrown on every beat and if one is caught, the same ball is thrown.

We will only cover so-called *open-hand juggling*. In this setting, the end effector is unactuated and funnel-shaped, meaning the ball is only held in place by stiction and gravity.

Given this juggling formulation, we explore a variety of semi-simple sequences with increasing complexity and the transition between them. Non-stationary wind is added to the environment to test the robustness of the proposed planning algorithm.

1.2. Contributions

This work builds upon a simulation environment and planning algorithm by Ploeger et al. [15]. Our main contributions are:

1. A computational implementation of a siteswap graph, including algorithms to generate a sequence of throws to juggle a variety of patterns and the transitions between them.
2. The identification of previously unreported trajectory constraints for open-handed toss juggling.
3. Grasp-free contact management between the ball and funnel-shaped end effector.
4. An estimator for air movement based on the flight path of the juggled balls.
5. A control algorithm that adapts to ball trajectories and wind estimates to allow juggling of many juggling patterns.

Upon this basis, further contributions can be made to the field of dynamic manipulation tasks, especially in the context of contact manipulation and robotic juggling.

2. Background and Foundations

Juggling has a long-standing tradition in human history. The earliest depictions of juggling can be found found in the Beni Hassan tomb, dated 1994-1781 B.C. Those depictions show women juggling with up to three balls [6]. There are accounts of sword juggling in the Chinese Book of Lie Zie written between 475-221 B.C. [17]. The Talmud even mentions torch juggling. Further, there were so-called *joculatores*, who entertained the people of the Roman Empire as traveling artists. They performed juggling and other slide-of-hand tricks [11].

In modern times, the advent of the circus made juggling more popular. Since the mid 20^{th} century juggling as a hobby has seen a significant spike in interest resulting in the establishment of multiple juggling organizations. Claude Shannon, the father of information theory, was a member of the MIT juggling club and produced important foundational work for juggling in academia. With the emergence of the internet and the possibility of international communication, standardized terminology, and notation have been developed, providing the basis for a thorough mathematical inquiry into the topic.

With such a rich history and many modern practitioners, it is no surprise that juggling has been studied rigorously from a mathematical perspective. Moreover, roboticists are working on reproducing human juggling with a robot.

2.1. Mathematics of Juggling

The theory of juggling has been studied extensively. In addition to dealing with historical aspects of juggling, Shannon [26] defined juggling terminology and described fundamental connections between the number of hands, number of balls, flight duration, and hand movement timing. He also conducted measurements of these quantities in human experiments. Buhler et al. [3] describe the mathematical properties of siteswap juggling. In his

book on juggling theory Burkhard Polster [16] provides a comprehensive formalization of the theory of juggling in general, including single and multi-handed juggling, siteswap juggling, and multiplexing.

2.1.1. Terminology

Two broad types of juggling can be differentiated: Toss and paddle juggling. In toss juggling, a ball is held for a non-zero amount of time after it is caught. This time is called dwell duration T_{dwell}. During the vacant duration T_{vacant} the hand is empty. The cycle duration $T_{\mathrm{cycle}} = T_{\mathrm{dwell}} + T_{\mathrm{vacant}}$ is the time from throw to throw. In between the throws one ball is caught.

The number of balls is denoted by N_{balls}, and the number of hands is denoted by N_{hands}. We always assume two hands throughout this work. The time a ball spends in the air is the flight duration T_{flight}. Note that this number can differ from ball to ball depending on the juggled pattern.

The limit of $T_{\mathrm{dwell}} \to 0$ is called paddle juggling, in which the ball is never held but only bounces off the end effector. Paddle juggling has been extensively discussed in the literature but has received little interest in the juggling community outside academia. This limit case will not be considered in the following since this work focuses on toss juggling since uniform juggling patterns (section 2.1.2) and siteswap juggling patterns (section 2.2) are only possible in toss juggling. This is because in paddle juggling the dwell duration T_{dwell} equals 0. Therefore, the distance the ball travels in the hand is zero. Consequently, the position from which the ball takes off is the same as the landing position of the ball. This leads to collisions making the previously mentioned juggling patterns impossible.

Considering the phase shift between both hands, an additional classification between juggling techniques is made. In synchronous juggling, both hands catch and throw at the same time, while in asynchronous juggling, the hand phases are shifted around half a cycle duration - one hand catches when the other throws. In this work, we only consider asynchronous juggling with $T_{\mathrm{dwell}} = T_{\mathrm{vacant}}$ and a phase shift of $T_{\mathrm{cycle}}/2$ between both hands.

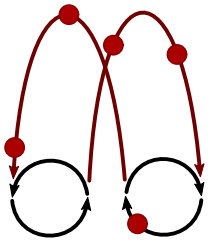
We will only consider a constant throwing position for each hand respectively. The catching position for each hand is also assumed to be constant, although it may be adapted to disturbances in the ball trajectory if necessary.

2.1.2. Uniform Juggling

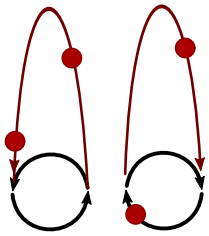
In uniform juggling, all flight durations T_{flight} are equal. Consequently, all balls are thrown to the same height since the flight time exclusively determines the throw height. In this scenario, the following equation [26] holds.

$$\frac{T_{\text{flight}} + T_{\text{dwell}}}{T_{\text{vacant}} + T_{\text{dwell}}} = \frac{N_{\text{balls}}}{N_{\text{hands}}}$$

Given the assumptions made in section 2.1.1 uniform juggling with an odd number of balls is called a cascade, while juggling uniformly with an even number of balls is called a fountain. In fountain patterns, balls never switch hands, while in cascades, balls switch hands every throw. This is necessary because the odd number of balls cannot be evenly distributed among both hands. Asynchronously juggling a fountain pattern with an odd number of balls would require different flight times of balls juggled in the left and right hands, respectively.



(a) Frontal view of a 5 ball cascade [15].



(b) Frontal view of a 4 ball fountain [15].

Figure 2.1.: Frontal view of cascade and fountain pattern. Balls switch hands at each throw in a cascade pattern, while a fountain pattern is characterized by balls never switching hands

2.1.3. Theoretical Limitations on Ball Number

For cascade patterns, a theoretical limit can be derived [15]. The horizontal velocity of the balls can be assumed constant. Therefore, the maximum number of balls that can be juggled in cascade is limited by the distance between the catching and throwing positions, as well as the ball radii, since the balls have to move past each other during the trajectory: Consequently, the inequality

$$N_{\text{balls}} < \left\lfloor \frac{d}{r} + 2\frac{T_{\text{dwell}}}{T_{\text{cycle}}} \right\rfloor$$

with the horizontal distances between the throw and catch position d and the ball radius r holds.

The number of balls is limited by the number of balls that can fit next to each other between throw and catch-position (d/r). Holding the balls longer in relation to having the hands empty ($2T_{\text{dwell}}/T_{\text{cycle}}$) increases the theoretically maximal number of balls.

2.2. Siteswap Juggling

In classical uniform juggling, all balls are thrown to the same height. Because of this, a uniform pattern is completly defined by the number of balls. To describe juggling patterns with throws of different heights, the siteswap notation is widely used. A sequence of integers $s = (s_1, s_2, \ldots, s_l)$ defines a siteswap pattern. Every element of the sequence represents one throw (also called height). A ball that is thrown as a j-throw is caught j catches later. For simplicity, the mathematical notation is often omitted and the sequence is written as a string of numbers, e.g. 744 instead of $(7, 4, 4)$. To execute a siteswap pattern the throws of the pattern are executed in succession, alternating the hands between each throw. After executing the last throw in a sequence, the juggler will continue with the first throw, repeating the sequence. See fig. 2.2 for trajectories of different throws. Classical cascade Juggling is a subset of site-swap patterns. For example, a five-ball cascade can be described by the siteswap pattern 5 (fig. 2.3). A visualisation of the siteswap pattern 744 in which five balls are juggled at different throwing heights is provided in fig. 2.4b).

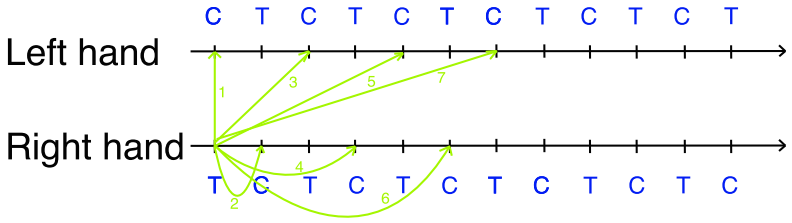
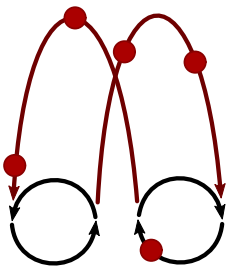
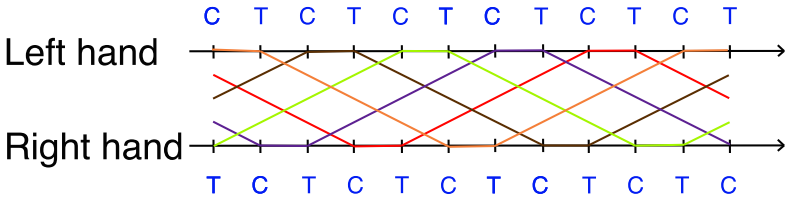


Figure 2.2.: The chart shows when and where which throw is caught. **C** denotes a catch and **T** denotes a throw. Throws 0, 1, and 2 are special cases. A 0-throw represents a beat at which no ball is thrown because the hand is empty. 1 corresponds to passing the ball between the hands without flight time. A 2-throw is a ball thrown and immediately caught again by the same hand. In practice, it does not leave the hand but is held until its next real throw.

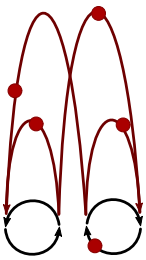


(a) Frontal view of 5 pattern with five balls [15].

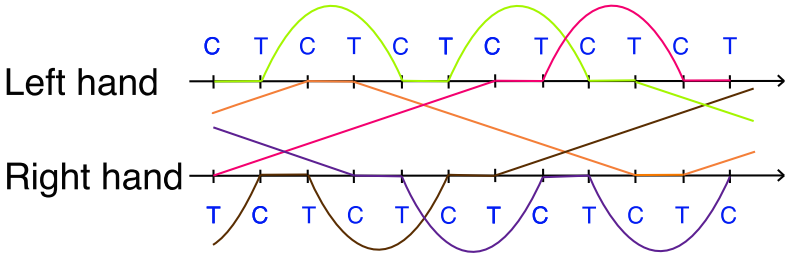


(b) Timing chart for 5 pattern with five balls. Each colored line represents the trajectory of one ball. **C** denotes a catch and **T** denotes a throw.

Figure 2.3.



(a) Frontal view of 744 pattern with five balls [15](adapted).



(b) Timing chart for 744 pattern with five balls. Each colored line represents the trajectory of one ball. **C** denotes a catch and **T** denotes a throw.

Figure 2.4.

Jugglability of a siteswap pattern

Nevertheless, not all finite sequences of integers correspond to valid juggling patterns. A quick test to check whether a sequence of integers can be juggled is provided by the *Average Theorem* [16]. Given a valid juggling sequence $s = (a_1, a_2, ..., a_l)$ that respects assumption **(**J3) with only one ball in hand at maximum, the number of balls that are needed to juggle is equal to its average.

$$N_{\text{balls}}(s) = \frac{1}{l}\sum_{i=1}^{l} a_i$$

For a sequence to be jugglable, its average must be an integer.

To fully verify that a given sequence is jugglable, the *Permutation Test* [16] can be performed. The test checks whether two throws in the sequence land on the same beat of the pattern. If this is not the case, at most one ball is caught per beat. Hence, the pattern is jugglable. This intuition translates to the following theorem. A sequence $s = (a_1, a_2, ..., a_l)$ is jugglable if and only if the function

$$\begin{aligned} \Phi_s : &\mathbb{Z}_l \to \mathbb{Z}_l \\ &i \mapsto a_i + i \mod l \end{aligned}$$

is a bijection of $\mathbb{Z}_l$ to $\mathbb{Z}_l$ (a permutation of $\mathbb{Z}_l$). $a_i + i \mod l$ is the beat at which the i^{th} throw lands. If Φ_s is a permutation, Φ_s maps each throw to a different landing beat, which means that at most one ball is caught in one beat.

Checking whether a sequence is jugglable can also be done by finding the corresponding cycle in a *siteswap graph*. A more rigorous mathematical definition will follow in the next section. Intuitively this can be understood as testing if the execution of the juggling sequence will return to the same ball state as before its execution. In this case, the execution can be repeated periodically and is jugglable.

2.2.1. Siteswap Graph

An alternative representation of site swap juggling patterns is a siteswap graph. Each state represents the catches to come at successive beats. A 1 represents a catch, while a 0 represents an empty beat. For example, the state string 11010 means that in the coming two beats, a ball will be caught, followed by an empty beat and then another ball catch. Transitions are labeled with the height of the executed throw. Transitions operate on the state with a left shift followed by a substitution. The left shift corresponds to the passing of time since the first beat in the string passed. The substitution is required since the thrown ball is not removed from the system but thrown so that it will be caught at a later beat. Consequently, a zero in the shifted state string is replaced by a one at the position indicated by the throw height. Note that a throw can only be performed if the beat at which it lands is not already marked with a 1 since this would result in catching multiple balls at once and violatating the assumption **J3** of *semi-simple juggling*.

An exception to the shift-and-substitute characteristic of state transitions is the 0-throw. It is performed if and only if the first character of the state is a 0. In this case, no ball is caught to be thrown. The state string is only shifted, but no substitution occurs since no ball is rethrown. Technically, a 0-throw is no throw at all.

The siteswap graph is defined by $\mathcal{M} = (V, E, \delta)$ with states V, throw heights E and transition relation δ (eq. (2.1)). The height at which the balls can be thrown is bound by

$h_{\max}$.

$$\mathcal{M} = (V, E, \delta) \tag{2.1}$$

$V = \{x \in \{0,1\}^{h_{max}} | \text{checksum}(x) = N_{\text{balls}}\}$ (the state always includes exactly N_{balls} balls)

$E = \{0, 1, ..., h_{max}\}$

$\delta = \{(x, h, (x << 1)[h \to 1]) |$ (Transition from state to shifted and substituted state)

$x \in S$

$\wedge\, x_1 = 1 \wedge (x << 1)_h = 0\}$ (A ball can only be thrown if one has been caught and no other catch is happening in the planned touch-down beat)

$\wedge\, 1 \leq h \leq h_{max}$ (h has to be an allowed throw.)

$\cup$

$\{(x, 0, (x << 1)) |$ (With a zero throw the state is only shifted)

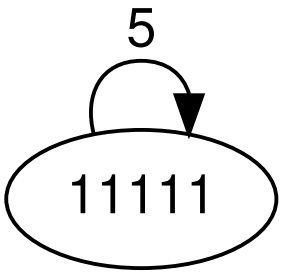
$x \in S$

$\wedge\, x_1 = 0\}$ (Zero throws are only allowed if no ball has been caught)

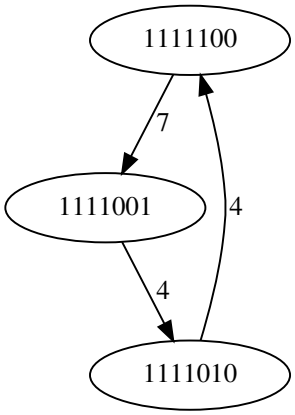
The state description includes information for both hands. To acquire information about only one hand, every other character of the description string is relevant. This implies that even throws are always caught by the hand they were thrown from, while odd throws are caught by the other hand.

We call the state which is represented by N_{balls} ones followed by $h_{\max} - N_{\text{balls}}$ zeros the ground state. All other states in the graph are called excited states.

In general, a siteswap pattern is represented by a cycle in the siteswap graph. All possible siteswap patterns for a given number of balls and maximal throw height can be determined by defining the full graph and searching for cycles. A complete graph for five balls and a throw height of 7 can be found in appendix A. Reduced subsets of the graph can be seen in fig. 2.5a and fig. 2.5b depicting a 5-ball-cascade and the 744-pattern respectively.



(a) Depiction of the graph of a 5-ball cascade.



(b) Depiction of the graph of a 744 siteswap pattern.

Figure 2.5.: Both figures show cycles in a siteswap graph corresponding to a siteswap pattern. The cycle for the 5 ball cascade is just a reflexive transition from and to the ground state since this pattern is uniform and the abstract ball state does not change. The cycle of the 744-siteswap includes multiple nodes because the pattern passes through three distinct ball states before repeating.

2.3. Robotic Juggling

2.3.1. Robotic Paddle Juggling

Paddle juggling is a well-studied topic in the robotics community, partly because of the well understood contact dynamics. The contact between the ball and the paddle can be modeled as an instantaneous impulse exchange.

Schaal et al. [24], [25] laid the foundational groundwork for open-loop paddle juggling. They describe a ball bounding on a planar end effector with one acuated degree of freedom along its normal. Schaal et al. showed the stability of periodic bouncing to the same height when the system is actuated with sinusoidal motion. This stability arises from the fact that the plate is decelerating when the ball hits it. A ball that peaks at a lower height hits the plate earlier when the plate still has a higher velocity. Analogously, a ball that has

been thrown higher hits the plate later and consequently bounces less. Reist et al. [18] also stabilize the system horizontally using a parabolic plate.

In the closed loop regime, Buhler et al. [4] proposed the so called mirror law. Under this control scheme, the paddle end effector mirrors the ball's movement along the symmetry plane given by $z = h_{\text{contact}}$. The mirroring might be distorted, resulting in the following relation between the end effector position x and the ball position b along the z-axis: $x - h_{\text{contact}} = \kappa(b - h_{\text{contact}})$. By adapting κ, the ball can be bounced periodically to arbitrary heights. Based on the mirror law Rizzi et al. [21], [20] used a three DOF robot to paddle-juggle two balls. The juggling robot can be seen in this video.

2.3.2. Robotic Toss Juggling

One of the first toss juggling machines was proposed by Claude Shannon. The system consists of one pole with a basket at each end. It is actuated by a motor that rotates the pole. Below the robotic arm is a bouncy drum. When a sinusoidal motion is applied to the central joint, the ball is thrown out of one basket, bounces off the drum, and jumps into the opposite basket. Schaal et al. [24] studied the system and could juggle up to five balls. A picture of the system can be seen in fig. 2.6.

Figure 2.6.: Claude Shannon's juggling machine. Image taken from the CMU robotics archive.

Kizaki et al. [9] proposed a robotic arm with a gripper as an end effector that can juggle two balls simultaneously. The hand movement is split into a throwing, catching, and returning phase. In the throwing phase, a movement primitive is followed from the return

to the throwing position. For catching, the inverse kinematics problem of the ball position at touch down is solved. Joint trajectories for the catching and return phase are generated by third-order polynomial interpolations between throwing and catching, and catching and return positions, respectively. The authors also implemented a re-grasping motion in the robotic gripper during dwell duration to increase grasp stability.

Sakaguchi et al. first analyzed the hand trajectories of human jugglers and modeled them as parametrized elliptic end effector trajectories [22]. Based on that work, they archived juggling with two balls on a two DOF robotic arm with a funnel-shaped end effector. They employed a hierarchical learning algorithm to adapt the parameters of the elliptical movement [23].

Riley et al. [19] presented a closed-loop algorithm using programmable pattern generators. They applied it to a humanoid robot with a baseball glove as an end effector that could catch incoming balls. Much like ours, their robot also hit the ball unintentionally before the intended catch. We discuss our solution to this problem in section 3.3.3. Kober et al. [10] were able to archive similar catching behavior for a humanoid robot. Their system can also throw the balls back to the interacting human. They could archive three-ball human-robot partner juggling by speeding up the time between catching and throwing.

Two-dimensional simplifications of robotic toss juggling have been studied in simulation and on physical systems. Chemin et al. [5] proposed a two-dimensional siteswap juggling model learning catching and throwing movements with reinforcement learning. Barsa et al. [1] build a siteswap juggling system on an inclined plane. This system uses fixed spring loaded shooting devices as hands. Marbles are accelerated onto the plane, rolling into the barrel of the other hand respectively. Both approaches neglect the complex contact dynamics of juggling: Chemin assumes the balls to be fixed to the corresponding hand during dwell duration, while Barsa does not accelerate the balls using hand movement but with dedicated devices.

Ploeger et al. [14] proposed an approach to learning two-ball one-handed juggling using reinforcement learning. They successfully demonstrated their open-loop algorithm on a 4 DOF manipulator. Besides the learned system, Ploeger et al. [15] also presented a planning-based closed-loop approach that juggles up to a 17-ball cascade in simulation. This system is the basis of this work.

2.4. Wind and Air Drag

Air drag affects an object by applying a force in the direction of the relative speed between the air and the body. The magnitude of the force is proportional to the square of the relative speed and described by the drag equation

$$F_D = \frac{1}{2}\rho u^2 c_d A$$

with the drag force F_D, the air density ρ, the relative velocity $u = \dot{x}_{\text{air}} - \dot{x}_{\text{body}}$ between air and body, the drag coefficient c_d and the cross-sectional area A in the direction of the relative velocity. Since the velocities are three-dimensional in our setting, the drag equation has to be adjusted using the vector norm to the form

$$F_D = \frac{1}{2}\rho c_d A \|\boldsymbol{u}\|_2 \boldsymbol{u}. \tag{2.2}$$

Including air drag, the ball movement is then described by

$$m_{\text{ball}}\ddot{\boldsymbol{x}}_{\boldsymbol{ball}} = \frac{1}{2}\rho c_d A \|\boldsymbol{w} - \dot{\boldsymbol{b}}\|_2 (\boldsymbol{w} - \dot{\boldsymbol{b}}) + m_{\text{ball}}\boldsymbol{g} \tag{2.3}$$

with ball mass m_{ball}, gravitational constant $\boldsymbol{g}$ and relative velocity $\boldsymbol{w} - \dot{\boldsymbol{b}}$.

Note that lift forces based on the Magnus effect are ignored since ball rotation is negligible. For further information on the interaction of flying balls with air, see [7] or [2].

Wind Estimation

In the scope of this work, the wind has to be estimated based on observable variables to throw balls accordingly to reach the desired target. A similar task has to be solved in drone flight: A drone has to reach a given target in a windy environment without prior knowledge of the wind conditions. For that purpose, Stepanyan et al [27] proposed an estimator system, that accurately estimates wind in an urban environment only using the prediction error of the drones' dynamics as an input.

2.5. Trajectory Optimization

We employ trajectory optimization [8] to plan when the robot has to move where in joint space, i.e. joint state trajectories. It has to be differentiated from dynamic programming, which also solves the optimal control problem and provides a complete policy instead of just a trajectory. Further, trajectory optimization approaches can be divided into *collocation* and *shooting* methods, which differ in the applied numerical integration method.

Shooting uses explicit integration schemes. It optimizes only with respect to the control signals, simulating the trajectory forward in time and optimizing on the backward pass. Therefore the system's dynamics are included in the objective function of the resulting optimization problem

$$\begin{aligned} &\min_{\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_K} c(\boldsymbol{x}_0, \boldsymbol{u}_0) + c(\boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{u}_0), \boldsymbol{u}_1) + c(\boldsymbol{f}(\boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{u}_0), \boldsymbol{u}_1), \boldsymbol{u}_2) + \ldots \\ = &\min_{\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_K} c(\boldsymbol{x}_0, \boldsymbol{u}_0) + \sum_{k=1}^{K} c(\boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}), \boldsymbol{u}_k) \end{aligned}$$

with states $\boldsymbol{x}_k$, control signals $\boldsymbol{u}_k$, system dynamics $\boldsymbol{f}$ and cost function c.

Collocation, on the other hand, employs implicit integration. It optimizes not only over the control signals but also the states. To enforce the system's dynamics so-called *collocation constraints* are added to the optimization formulation.

$$\begin{aligned} &\min_{\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \ldots, \boldsymbol{x}_K, \boldsymbol{u}_K} \sum_{k=0}^{K} c(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ &s.t. \forall k : \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) = \boldsymbol{x}_{k+1}. \end{aligned}$$

Trajectory optimization can be subdivided further regarding the discretization of continuous dynamics. Although the distinction is not strict, the *h-method* typically uses lower order polynomials with a high number of interpolation points, while the *p-method* uses higher order polynomials with fewer interpolation points.

2.6. Optimization via the Interior Point Method

One way of solving constrained optimization problems is to combine the objective function and the constraints into one surrogate function to be minimized. If such a function has

the same optimum as the original constrained objective function, finding the solution to the original problem breaks down to minimizing the corresponding function.

From now on the optimization problem

$$\begin{aligned} &\min_{x} f(\boldsymbol{x}) \\ &\text{s.t. } c_j(\boldsymbol{x}) \geq 0 \text{ for } j = 1, ..., l \end{aligned}$$

will be discussed. Note that this formulation does not violate generality since equality constraints can be rewritten as two inequality constraints.

The Interior Point Method [13], [30], [28] constructs a so-called Barrier Function from the given objective function and constraints.

$$B(\boldsymbol{x}, \boldsymbol{\mu}) = f(\boldsymbol{x}) - \boldsymbol{\mu} \sum_{j=1}^{l} \ln c_j(\boldsymbol{x})$$

Since $\ln c_j(x)$ converges to $-\infty$ for $c_j(x) \to 0$, the latter term functions like a barrier for the possible values of x in the optimization problem. The solution of minimizing a series of barrier functions with decreasing $\boldsymbol{\mu}$ with respect to $\boldsymbol{x}$ while always staying within the feasibility region provided by the barrier converges to the solution of the initial constrained optimization problem.

To optimize joint space trajectories, this work will employ the shooting method with interior point solvers provided by IPOPT [29].

3. Methods

3.1. Overview of the Planning Workflow

The basis for juggling a sequence is an implementation of the *siteswap graph*, on which a *sequence planner* operates. The graph is traversed according to the *sequence planner*. Starting from the ground state, which corresponds to a uniform pattern, the system transitions to a desired pattern. Based on the graph traversal, the *sequence planner* provides a high-level command queue for each hand that includes the desired flight time and target of the next throw for that hand.

The *ball state dependent planner* operates on that queue. Since it also keeps track of the ball state, it combines information about the *environment* and the high-level command queue to produce throwing and catching positions, velocities, and other constraining settings for the *low-level planner*. The *ball state dependent planner* is also informed by the *ball launcher* when another ball is added to the system to augment its internal estimate of the ball state. The *low-level planner* solves the trajectory optimization problem given the set of constraints provided by the *ball state dependent planner*. From the resulting reference trajectories, the *tracking controller* calculates joint torques that are executed in the simulation *environment*. The *ball launcher* can also affect the *environment* by adding a ball at a desired time step. The resulting *environment* state is then fed back to the *ball state dependent planner* to update its state estimate. Figure 3.1 shows the information flow between the different components of the system.

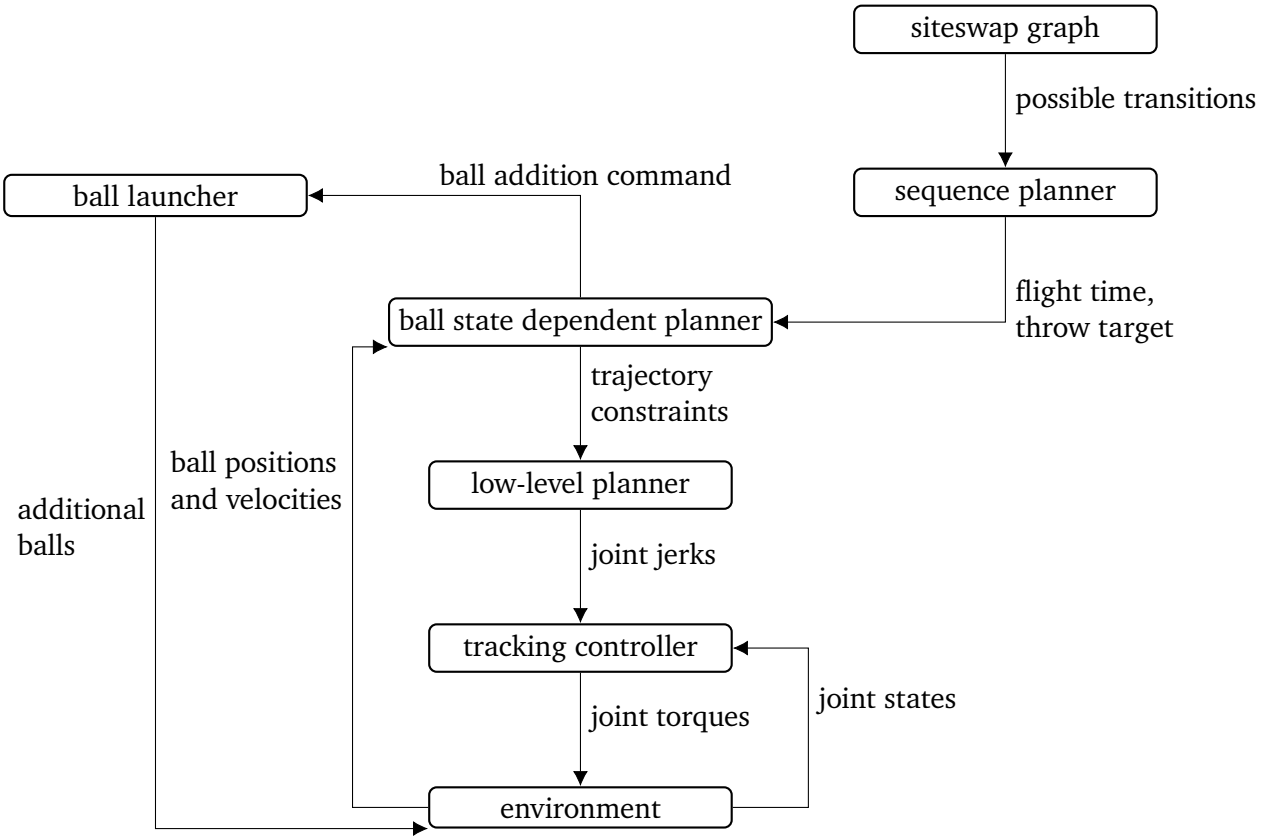


Figure 3.1.: Diagram of the system's control and information flow. The *siteswap graph* (section 3.2.1) includes all jugglable sequences. The *sequence planner* (section 3.2.3 - 3.2.4) selects a sequence, transforms it into flight times, and throw-targets for actual ball throws. From these, the *ball state dependent planner* (section 3.3) generates constraints for the *low-level planner* (section 3.4) which optimizes a trajectory in joint space. The *tracking controller* (section 3.5) converts the resulting joint jerks to joint torques, which are executed in the simulation *environment*. The *ball launcher* (section 3.2.5) adds additional balls to the *environment*, when needed and possible.

3.2. Planning Siteswap Patterns

3.2.1. Creating the Graph

As described in section 2.2.1, transitions in the siteswap graph can be viewed as operations on the state they are transitioning from. Since the graph is already iteratively defined in eq. (2.1), the graph can be generated iteratively using breadth-first-search. Generation starts from the ground state corresponding to the steady state of uniform juggling with N_{balls} balls. All transitions $0 \leq h \leq h_{\max}$ can be tested for every unchecked state. A transition is added to the graph if it produces valid states. Previously unseen states are also added to the graph. Once all unchecked states are checked, the algorithm terminates.

3.2.2. Finding the Pattern

A siteswap pattern is equivalent to a cycle in the siteswap graph. To be able to add balls easily our system always starts juggling from the ground state (see section 3.2.5). To juggle a pattern that does not include the ground state like 672, the corresponding cycle first has to be found and then the graph has to be traversed to the cycle. A depth-first search algorithm is usually used to find cycles in a graph. For this work, not all cycles are needed. It is enough to find a few specific patterns in the graph, for which we use the following algorithm.

Input: Siteswap graph $\mathcal{M} = (V, E, \delta)$, juggling sequence $s = (s_1, s_2, ..., s_l)$
Result: Sequence of states $\{v_1, v_2, ..., v_l\}$, that make up the sequence cycle or an empty sequence $()$ if no cycle is found.

```
for v ∈ V do
    for i ∈ 0, ..., |s| − 1 do
        j ← i;
        v′ ← v;
        cycle ← (v);
        while (v′, s_{j+1}, v″) ∈ δ do
            v′ ← v″;
            j ← j + 1 mod |s|;
            if j = i ∧ v′ = v then
                return cycle;
            end
            cycle ← cycle + (v′);
        end
    end
    return ();
end
```

Algorithm 1: The algorithm finds a cycle in the graph matching the pattern. For every node in the graph, it first checks whether there is an outgoing transition with one element of the sequence. If so, the algorithm follows the transition and tries to traverse consecutive transitions with elements in the sequence. If all elements in the sequence correspond to possible transitions and the starting node is reached with the last element of the sequence, the cycle corresponding to the sequence is found and can be returned.

This algorithm's runtime is linear in the number of states and has a worst-case computational complexity of $O(|V| \cdot |s|^2)$. In reality, following a potential cycle match is rarely necessary, which reduces complexity to $O(|V| + \epsilon|V|)$ with small ϵ. Therefore, checking for a specific cycle in the graph is efficiently possible.

3.2.3. Navigation to the Pattern

Initially, the system is always brought into the ground state, corresponding to a uniform cascade or fountain pattern, as detailed in section 3.2.5. From here, the siteswap graph has to be traversed from the node representing the uniform pattern to one node included in the

pattern cycle. The Dijkstra algorithm that could have been applied here is overly complex since all edges have equal weight. Consequently, we employed the simpler breadth-first search algorithm. Starting from the uniform pattern node nodes with successively higher hop counts from the uniform node are traversed. Once a node from the sequence cycle is reached, the path to the cycle is found.

The final throw sequence includes several uniform throws as a startup sequence, the transitional path, and finally an arbitrary number of repetitions of the desired juggling sequence.

3.2.4. Translating Action Sequences to Flight durations and Targets

To juggle a given pattern on the robot the integers in the juggling sequence have to be converted to flight durations and target catching positions in the cartesian task space of the robot. From the flight duration, the target catching position, and the throwing position, the throw velocity can be calculated, which is then executed by the robot.

This work assumes that $T_{\text{dwell}} = T_{\text{vacant}}$ as well as a phase shift of $T_{\text{flight}}/2$ between the hands so that one hand catches at the same time, the other throws a ball.

As mentioned before, a 0 throw is no throw at all since no ball has landed to be thrown. A 1-throw represents a hand-over of the ball between the hands and has a zero flight duration. A 2-throw is thrown and caught again by the same hand after one vacant duration. In practice, jugglers usually do not throw a 2-throw and instead, just keep holding it in hand. Consequently, it also has a zero flight duration. Higher throws have flight durations that correspond to the number of dwell durations and vacant durations during the flight. The flight time is therefore determined by the expression

$$T_{\text{flight}}(a) = \begin{cases} 0, \text{for } a \leq 2 \\ T_{\text{vacant}} + (\frac{a}{2} - 1)T_{\text{cycle}}, \text{ for } a > 2 \text{ and } 2|a \\ (\frac{a-1}{2}T_{\text{cycle}}), \text{ for } a > 2 \text{ and } \neg\, 2|a \end{cases} \tag{3.1}$$

with throw action a. The expression makes a distinction between even and odd numbers. Odd numbers represent throws that land in the opposite hand, making it necessary to include the phase shift between the hands. This is not necessary for even numbers, because they correspond to throws to the same hand. For a graphical representation of different throws and their flight durations, see fig. 2.2.

3.2.5. Timing Ball Drops and Initial Sequence

If more than one ball is in the end effector at the same time, the balls cannot be thrown accurately. Consequently, juggling with more balls than hands requires the addition of more balls in the system while juggling. We always start with one ball in each hand, and the ball launcherdrops additional balls into the right hand. Drops into the right hand can only occur if no ball is in the right hand. The interval in which balls can be added is split into two parts: The first subinterval is before the ball from the left hand reaches the right hand, and a second after the original ball from the left leaves the right hand. The possible drop-in times for a 7-ball pattern are marked in fig. 3.2.

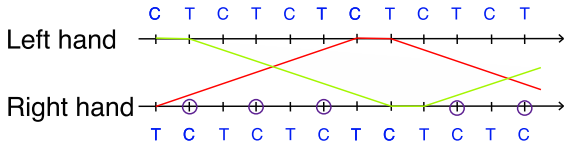


Figure 3.2.: Times at which a dropped ball can land for the 7-ball cascade are marked with purple circles. The interval in which the ball launcherdrops balls into the right hand is split in half by a 7-throw passing from the left side.

Consequently, the set

$$\Lambda_{\text{launch}} = \{T_{\text{vacant}} + (i-1)T_{\text{cycle}} - t_{\text{drop}} | i \in \mathbb{N}^+, \neg N_{\text{balls}} | i \wedge \neg N_{\text{balls}} | (i - \left\lceil \frac{N_{\text{balls}}}{2} \right\rceil)\}$$

includes all possible drop times. $t_{\text{drop}} = \sqrt{2h_{\text{fall}}/g}$ is the time it takes the ball to fall from the ball from the launcher into the hand. At the first $N_{\text{balls}} - 2$ times in Λ_{lauch} the ball launcher drops a ball. As mentioned, the *sequence planner* always includes a few repetitions of the uniform pattern to ensure enough time to add all necessary balls. Afterward, a transition to other patterns is possible.

3.3. Task Space Constraints

Planning the hand trajectories requires a series of constraints. Trivial ones include catching and throwing the ball at the correct position and velocity. More complex constraints are needed to ensure smooth separation and contact establishment between hand and ball as

well as to prevent premature ball contact during the vacant duration or ball rollout during the dwell duration.

Figure 3.3 provides a graphical overview of when constraints are applied. All constraints are described in detail in the following sections.

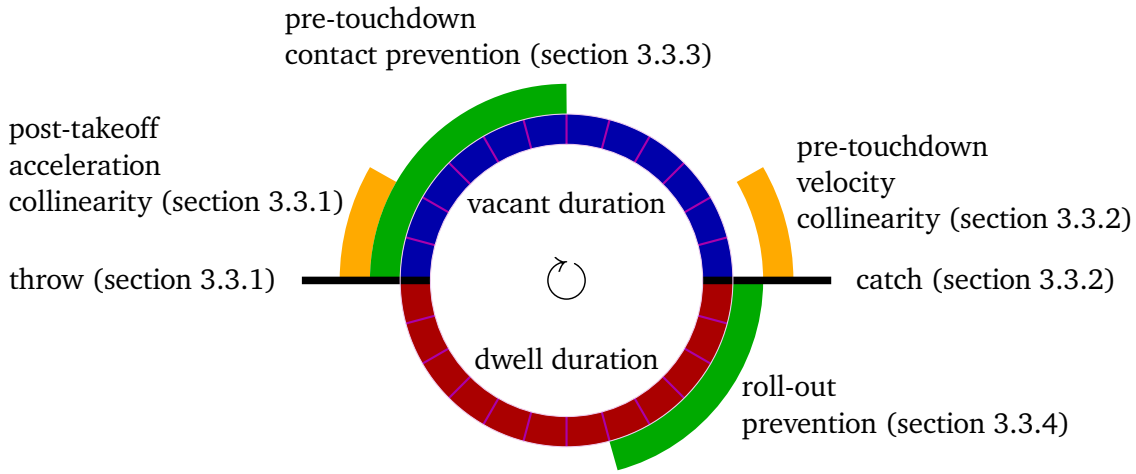


Figure 3.3.: Time intervals during which constraints are applied. Each segment in the inner ring represents one planning step.

The desired takeoff position of the balls $\boldsymbol{b}_{\mathrm{TO,des}}$ and the desired touchdown position $\boldsymbol{b}_{\mathrm{TD,des}}$ are user-choosen parameters. They determine the horizontal distance the ball has to be carried during dwell duration and the distance the ball has to fly during flight duration.

3.3.1. Throwing

To ensure that the ball reaches its target, the hand has to be at the correct position and velocity at takeoff time. The contact break is defined as the point where no forces act between hand and ball. To guarantee contact loss at takeoff, the hands' acceleration is constrained to equal the gravitational constant $\boldsymbol{g}$. The constraints are formulated as

$$\begin{aligned} \mathbf{0} &= \boldsymbol{x}(t_{\mathrm{TO}}) - \boldsymbol{b}_{\mathrm{TO,des}} \\ \mathbf{0} &= \dot{\boldsymbol{x}}(t_{\mathrm{TO}}) - \dot{\boldsymbol{b}}_{\mathrm{TO,des}} \\ \mathbf{0} &= \ddot{\boldsymbol{x}}(t_{\mathrm{TO}}) - \boldsymbol{g} \end{aligned} \tag{3.2}$$

with the predefined takeoff position $\boldsymbol{b}_{\mathbf{TO,des}}$ and hand position $\boldsymbol{x}(t)$. The desired takeoff velocity $\dot{\boldsymbol{b}}_{\mathrm{TO,des}}$ depends on the desired flight time, the target, and the wind conditions. If we neglect air drag, it is just the analytical solution to the quadratic equation of the ball trajectory and can be computed as $\dot{\boldsymbol{b}}_{\mathrm{TO,des}} = (b_{\mathrm{TD,des}} - b_{\mathrm{TO,des}} - 0.5gT_{\mathrm{flight}}{}^2)/T_{\mathrm{flight}}$. If we consider air drag, $\dot{\boldsymbol{b}}_{\mathrm{TO,des}}$ has to be computed numerically. This is described in detail in section 3.6.

To prevent redirection of the ball by the hand moving laterally right after takeoff, the relative acceleration between the hand and the ball has to be parallel to the hand normal $\boldsymbol{e}_h$ for a short interval after takeoff (fig. 3.4). The resulting constraint

$$\forall t \in [t_{TO}, t_{TO} + t_{\mathrm{post\_takeoff}}] : \ \mathbf{0} = (\ddot{\boldsymbol{x}}(t) - \boldsymbol{g}) \times \boldsymbol{e}_h(t) \tag{3.3}$$

will be referenced as *post takeoff constraint* in the rest of this work.

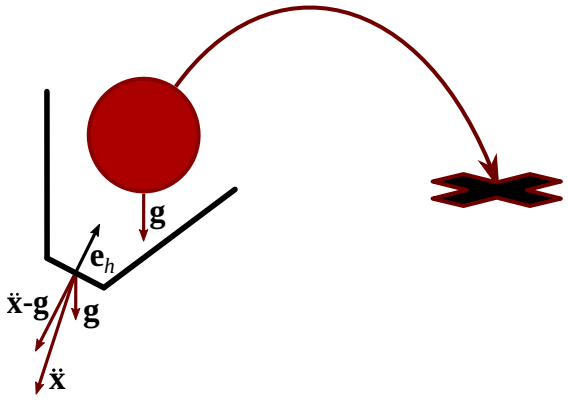


Figure 3.4.: This figure shows the ball and the end effector right after takeoff. The relative acceleration between the ball and hand has to be colinear for a short interval after takeoff to ensure proper separation. The figure was taken from [15].

3.3.2. Catching

For the ball to be caught and thrown correctly, the predicted position of the ball and the hand have to be equal to the ball's predicted touch-down time. Consequently,

$$\mathbf{0} = \boldsymbol{x}(\tilde{t}_{\mathrm{TD}}) - \tilde{\boldsymbol{b}}_{\mathrm{TD}} \tag{3.4}$$

has to hold with predicted touchdown time $\tilde{t}_{\mathrm{TD}}$ and predicted ball position at touchdown $\tilde{\boldsymbol{b}}_{\mathrm{TD}}$.

To further ensure that the ball lands smoothly in hand, the ball's touch-down velocity and the hands' velocity must be collinear in a short interval before touch-down (fig. 3.5). This is enforced by fulfilling the *pre touchdown constraint*

$$\forall t \in [\tilde{t}_{\mathrm{TD}} - T_{\mathrm{pre\_touchdown}}, \tilde{t}_{\mathrm{TD}}] : \ \mathbf{0} = \dot{\boldsymbol{x}}(t) \times \dot{\tilde{\boldsymbol{b}}}(t). \tag{3.5}$$

Originally, Ploeger et al. published both throwing and catching constraints in [15].

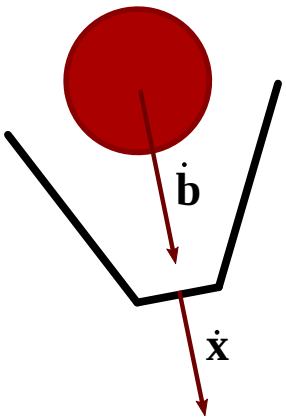


Figure 3.5.: This figure shows the ball and the end effector right before the touch-down. The ball and hand velocity must be collinear before touching down to ensure a smooth landing. The figure was taken from [15].

3.3.3. Pre Touchdown Contact Prevention

Some throws to the same hand, such as 4-throws, are thrown at a shallow height. When a ball is thrown subsequently to such a low throw, the hand will reach the same height

as the previously thrown ball. To catch the 4-ball, the hand has to move to the catching position again. Unconstrained, this will lead to the hand moving toward the ball, hitting it from the side. This leads to the redirection of the ball and prevents the catch.

The end effector's task space position must be constrained during the vacant duration to prevent hitting the ball before it is caught. The end effector should be at a safe distance from the ball during the vacant duration, except when the ball is directly above it because it must enter the cone to be caught. The resulting *pre touchdown contact prevention constraint* is formulated in eq. (3.6) with the funnels slope angle α, hand normal $\boldsymbol{e}_h$ and minimal distance to the hand $d_{\text{contact\_prevention}}$.

$$\forall t \in [t_{TO}, t_{TO} + T_{\text{vacant}}] : \; ||\boldsymbol{x}(t) - \boldsymbol{b}(t)|| > d_{\text{contact\_prevention}} \vee \measuredangle(\boldsymbol{b}(t) - \boldsymbol{x}(t), \boldsymbol{e}_h(t)) < \alpha \tag{3.6}$$

A visual representation of the constraint is shown in fig. 3.6.

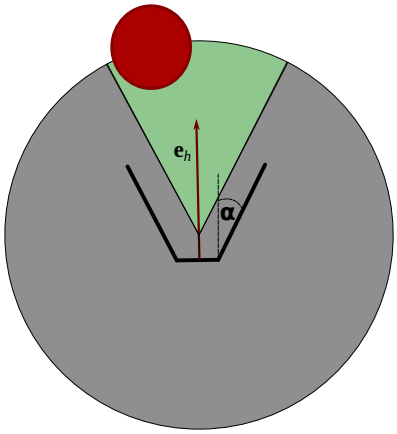


Figure 3.6.: Figure of Ball and end effector with funnel angle α during vacant duration. The *pre touchdown contact prevention constraint* is represented by the green and grey semi-circles. The grey area represents the forbidden volume defined by the first subterm, and the green area represents the allowed angle range formulated in the second subterm of eq. (3.6).

In practice, the constraint eq. (3.6) is overly complex. It can be simplified to a generic distance constraint by reducing the time it has to be fulfilled (eq. (3.7)). $t_{\text{contact\_prevention}}$

lies in the interval $[0, T_{\text{vacant}}]$, but is small enough so that the effector can get close when it is time to catch.

$$\forall t \in [t_{TO}, t_{TO} + T_{\text{contact\_prevention}}] : \ ||\boldsymbol{x}(t) - \boldsymbol{b}(t)|| > d_{\text{contact\_prevention}} \tag{3.7}$$

The resulting hand task-space trajectory is adjusted accordingly, as shown in fig. 3.7.

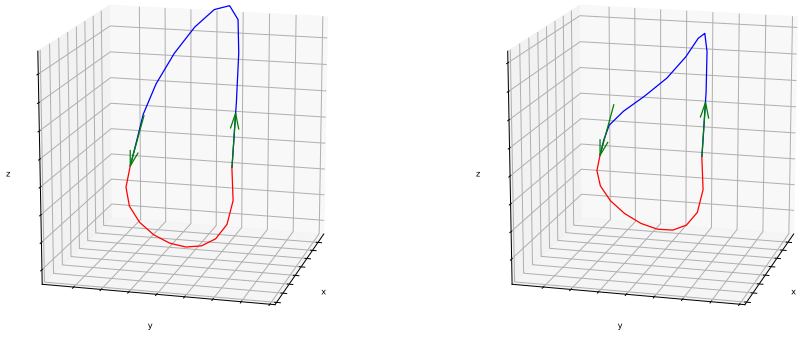


Figure 3.7.: End effector trajectory without (left) and with (right) *pre touch down contact prevention* constraint. The trajectory during dwell duration is marked in red; the trajectory during vacant duration is marked in blue. Green arrows show takeoff and touchdown and position and direction. The trajectory generated with the *pre touch down contact prevention* avoids the incoming ball by bending down.

3.3.4. Contact Management during Dwell Time

When juggling with open hands, the ball is not fixed to the hand but held in place by gravity and stiction. This is especially restricting when starting a throw from the resting

position with a ball in the hand since the initial velocity is zero. Consequently, it is necessary to constrain the effective acceleration of the ball relative to the end effector, such that it always rolls to or sits at the bottom of the cone. Enforcing the *rollout prevention constraint*

$$\forall t \in [T_{TD}, T_{TD} + T_{\text{roll\_out\_prevention}}] : \arccos\left(\frac{< \boldsymbol{e_h}(t), \boldsymbol{g} - \ddot{\boldsymbol{x}}(t) >}{||\boldsymbol{e_h}(t)||||\boldsymbol{g} - \ddot{\boldsymbol{x}}(t)||}\right) < 90° + \alpha \quad (3.8)$$

with the relative acceleration $\boldsymbol{g} - \ddot{\boldsymbol{x}}$ between hand and ball, the hand normal $\boldsymbol{e_h}$ and the funnel slope angle α prevents the ball from rolling out. For a graphical representation of acceleration, hand normals, and forbidden angles, see fig. 3.8.

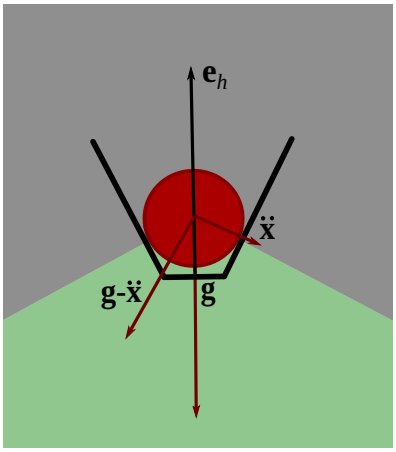


Figure 3.8.: This figure shows the funnel shaped end effector with normal e_h and ball. Accelerations g, $\ddot{x}$ and relative acceleration $g - \ddot{x}$ are marked in dark red. The grey area represents the forbidden angle range for the relative acceleration $g - \ddot{x}$,, the green area represents the allowed range as formulated in the *rollout prevention constraint* eq. (3.8).

Additional to constraining the acceleration during dwell time, the resting position can be adapted to facilitate contact management. In a pattern such as 672, a fast ball must be caught, stopped, and held for one cycle and afterward thrown to a very high altitude. By lowering the resting position to $10\,\text{cm}$ below the catching height, the ball does not stop abruptly, which prevents bouncing. As a side effect, the consecutive drawing motion can start from the lowest point of the movement, allowing a more stable contact before takeoff.

3.4. Planning in Joint Space

The actual trajectory optimization is performed in joint space. The optimization target is the minimization of the integral of the squared norm of the joint accelerations to generate a smooth trajectory. The constraints are formulated in joint space whenever possible by precomputing the inverse kinematics for the constraining value before optimization. Otherwise, forward kinematics are included in the constraint. The corresponding values can be precomputed in joint space for the throwing position, velocity, acceleration, and catching position. For all other constraints, differential forward kinematics are computed at optimization formulation. Unless stated otherwise the trajectory is planned for a complete cycle duration at once. Planning for catching and throwing an incoming ball is executed, at the moment the previous ball is thrown.

The complete optimization formulation formulated as

$$
\begin{aligned}
&\min_{\ddot{\boldsymbol{q}}(t)} \int_{t_{\mathrm{TD}}}^{t_{\mathrm{TD}}+T_{\mathrm{cycle}}} \ddot{\boldsymbol{q}}(t)^T \ddot{\boldsymbol{q}}(t) dt \\
\text{s.t. } &\mathbf{0} = \boldsymbol{q}(\tilde{t}_{\mathrm{TD}}) - \boldsymbol{q}_{\mathrm{catch}} && && \text{(section 3.3.2)} \\
&\mathbf{0} = \dot{\boldsymbol{f}}_{\mathrm{kin}}(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)) \times \dot{\tilde{\boldsymbol{b}}}(t) && \forall t \in [\tilde{t}_{\mathrm{TD}} - T_{\mathrm{pre\_touchdown}}, \tilde{t}_{\mathrm{TD}}] \\
&\mathbf{0} = \boldsymbol{q}(T_{\mathrm{cycle}}) - \boldsymbol{q}_{\mathrm{throw}} && && \text{(section 3.3.1)} \\
&\mathbf{0} = \dot{\boldsymbol{q}}(T_{\mathrm{cycle}}) - \dot{\boldsymbol{q}}_{\mathrm{throw}} \\
&\mathbf{0} = \ddot{\boldsymbol{x}}(t_{\mathrm{TO}}) - \ddot{\boldsymbol{q}}_{\mathrm{throw}} \\
&\mathbf{0} = (\ddot{\boldsymbol{f}}_{\mathrm{kin}}(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \boldsymbol{q}(t)) - \boldsymbol{g}) \times \boldsymbol{e}_h && \forall t \in [t_{TO}, t_{TO} + t_{\mathrm{post\_takeoff}}] \\
&d_{\mathrm{contact\_prevention}} < ||\boldsymbol{x}(t) - \boldsymbol{b}(t)|| && \forall t \in [t_{TO}, t_{TO} + T_{\mathrm{contact\_prevention}}] && \text{(section 3.3.3)} \\
&90^\circ + \alpha > \arccos\left(\frac{\boldsymbol{e}_h(t) \cdot (\boldsymbol{g} - \ddot{\boldsymbol{x}}(t))}{||\boldsymbol{e}_h(t)||||\boldsymbol{g} - \ddot{\boldsymbol{x}}(t)||}\right) && \forall t \in [t_{TO}, t_{TO} + T_{\mathrm{vacant}}] && \text{(section 3.3.4)}
\end{aligned}
\tag{3.9}
$$

with

$$\begin{aligned}
\boldsymbol{q}_{\text{catch}} =& \boldsymbol{f}_{\text{kin}}^{-1}(\tilde{\boldsymbol{b}}_{\text{TD}}) \\
\boldsymbol{q}_{throw} =& \boldsymbol{f}_{\text{kin}}^{-1}(\boldsymbol{b}_{\text{TO,des}}) \\
\dot{\boldsymbol{q}}_{\text{throw}} =& \dot{\boldsymbol{f}}_{\text{kin}}^{-1}(\boldsymbol{b}_{\text{TO,des}}, \dot{\boldsymbol{b}}_{\text{TO,des}}) \\
\ddot{\boldsymbol{q}}_{throw} =& \ddot{\boldsymbol{f}}_{\text{kin}}^{-1}(\boldsymbol{b}_{\text{TO,des}}, \dot{\boldsymbol{b}}_{\text{TO,des}}, \boldsymbol{g}).
\end{aligned}$$

The optimization problem is discretized using an explicit Euler integration scheme (eq. (3.10)) with $\Delta t = 0.02\,\text{s}$.

$$\begin{aligned}
\boldsymbol{q}_k &= \boldsymbol{q}_{k-1} + \dot{\boldsymbol{q}}_{k-1}\Delta t + \frac{1}{2}\ddot{\boldsymbol{q}}_{k-1}\Delta t^2 + \frac{1}{6}\dddot{\boldsymbol{q}}_{k-1}\Delta t^3 \\
\dot{\boldsymbol{q}}_k &= \dot{\boldsymbol{q}}_{k-1} + \ddot{\boldsymbol{q}}_{k-1}\Delta t + \frac{1}{2}\dddot{\boldsymbol{q}}_{k-1}\Delta t^2 \\
\ddot{\boldsymbol{q}}_k &= \ddot{\boldsymbol{q}}_{k-1} + \dddot{\boldsymbol{q}}_{k-1}\Delta t
\end{aligned} \tag{3.10}$$

Using methods from IPOPT and the ma27 linear solver, the optimization problem is then solved during runtime. The integration scheme was taken from [15].

3.5. Joint Level Control

For computing the joint torques $\boldsymbol{\tau}$ an inverse dynamics control law

$$\begin{aligned}
\ddot{\boldsymbol{q}}_{\text{ref}} =& \ddot{\boldsymbol{q}}_{\text{d}} + \boldsymbol{K}_P(\boldsymbol{q_d} - \boldsymbol{q}) + \boldsymbol{K}_D(\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}) \\
\boldsymbol{\tau} =& \boldsymbol{f}_{\text{dyn}}^{-1}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}_{\text{ref}})
\end{aligned}$$

with desired joint position $\boldsymbol{q}_d$, desired joint velocity $\dot{\boldsymbol{q}}_d$, desired acceleration $\ddot{\boldsymbol{q}}_d$ is used. The desired joint states are interpolated from the results of the joint space optimization in section 3.4. The inverse dynamics model $\boldsymbol{f}_{\text{dyn}}^{-1}$ is adjusted for ball mass if the ball is in hand. We employ the controller with high gains, leading to negligible tracking errors in simulation.

3.6. Adapting to Wind

Because of the complex nature of the dynamical equations, including drag, the wind equations were solved numerically in this work. Therefore, the planning of takeoff velocities and the prediction of touch-down time, position, and velocity had to be done numerically. To adapt to non-stationary wind, the current air velocity is estimated as described in section 3.6.2.

3.6.1. Adaptive Throwing

To reach a desired target position $\boldsymbol{b}_{\mathrm{TD}}$ with a desired flight duration T_{flight}, the following optimization problem in eq. (3.11) has to be solved.

$$\begin{aligned} &\min_{\dot{\boldsymbol{b}}_{\mathrm{TO}}} \left\| \boldsymbol{b}_{\mathrm{TO}} + \int_{t_{\mathrm{TO}}}^{t_{\mathrm{TO}}+T_{\mathrm{flight}}} \dot{\boldsymbol{b}}_{\mathrm{TO}} + \left(\int \ddot{\boldsymbol{b}}(t) dt \right) dt - \boldsymbol{b}_{\mathrm{TD}} \right\| \\ &\text{s.t.} \quad \ddot{\boldsymbol{b}}(t) = \frac{1}{m_{\mathrm{ball}}} \boldsymbol{F}_D(\boldsymbol{w}, \dot{\boldsymbol{b}}(t)) + \boldsymbol{g} \end{aligned} \tag{3.11}$$

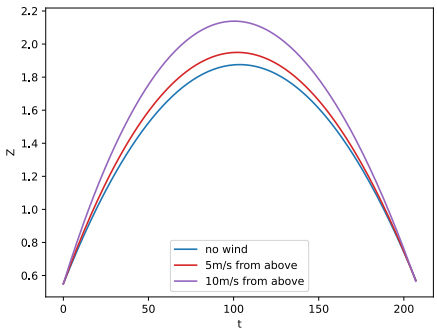
In practice, this optimization is unstable, and the solution tends to diverge. Diverging behavior can be combated by introducing the idealized takeoff velocity $\dot{\boldsymbol{b}}_{\mathrm{TO,ideal}}$ (the velocity at which the ball would have to be thrown without the influence of air) as a prior. The distance from this idealized velocity is chosen as the target function, moving the objective of reaching the target to the constraints section of the problem. This slightly unintuitive but better-behaved optimization problem is defined in eq. (3.12).

$$\begin{aligned} &\min_{\dot{\boldsymbol{b}}_{\mathrm{TO}}} \left\| \dot{\boldsymbol{b}}_{\mathrm{TO}} - \dot{\boldsymbol{b}}_{\mathrm{TO,ideal}} \right\| \\ &\text{s.t. } 0 = \left\| \boldsymbol{b}_{\mathrm{TO}} + \int_{t_{\mathrm{TO}}}^{t_{\mathrm{TO}}+T_{\mathrm{flight}}} \dot{\boldsymbol{b}}_{\mathrm{TO}} + \left(\int \ddot{\boldsymbol{b}}(t) dt \right) dt - \boldsymbol{b}_{\mathrm{TD}} \right\| \\ &\ddot{\boldsymbol{b}}(t) = \frac{1}{m_{\mathrm{ball}}} \boldsymbol{F}_D(\boldsymbol{w}, \dot{\boldsymbol{b}}(t)) + \boldsymbol{g} \end{aligned} \tag{3.12}$$

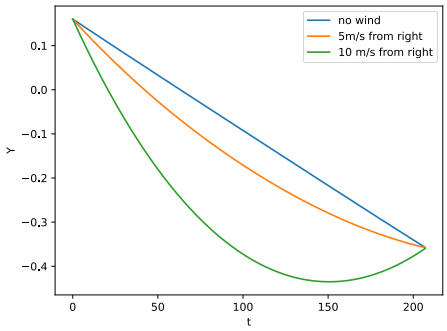
The optimization problem is discretized using the following explicit Euler integration scheme eq. (3.13) with $\Delta t = 0.005\,\mathrm{s}$.

$$\begin{aligned} \boldsymbol{b}_k &= \boldsymbol{b}_{k-1} + \dot{\boldsymbol{b}}_{k-1}\Delta t + \frac{1}{2}\ddot{\boldsymbol{b}}_{k-1}\Delta t^2 \\ \dot{\boldsymbol{b}}_k &= \dot{\boldsymbol{b}}_{k-1} + \ddot{\boldsymbol{b}}_{k-1}\Delta t \end{aligned} \tag{3.13}$$

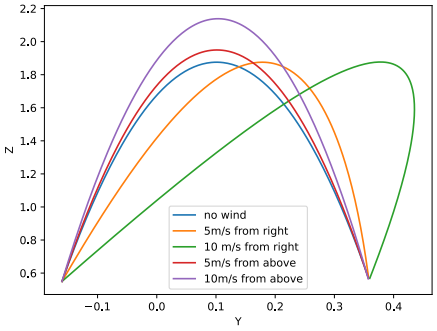
Using methods from IPOPT and the ma27 linear solver, this optimization problem is solved during runtime. The resulting takeoff velocity $\dot{\boldsymbol{b}}_{\mathrm{TO}}$ is chosen to compensate for the drag force and reach the target. The resulting trajectories can be seen in Figure 3.9.



(a) Height of ball trajectories over time subjected to wind from above



(b) Y-component of ball trajectories over time subjected to side wind



(c) Ball trajectories in YZ-phase space subjected to different winds

Figure 3.9.: Ball trajectories with computed takeoff velocity subjected to different winds. The balls all reach the same desired target.

The ball dynamics (eq. (2.3)) are numerically integrated using the integration scheme in eq. (3.13) to predict the touch-down time, position, and velocity. This is necessary to find the position to catch the incoming ball and fulfill the touch-down collinearity constraint mentioned previously.

3.6.2. Wind Speed Estimation

A state estimator system estimates air movement independent of prior wind speed and direction knowledge. The system is defined in eq. (3.14) with the observed position of the i^{th} ball $\boldsymbol{y}_i$.

$$\begin{aligned}
\dot{\hat{\boldsymbol{b}}}_{\boldsymbol{i}} &= \hat{\boldsymbol{b}}_i + \boldsymbol{K}_{pos}(\boldsymbol{y}_i - \hat{\boldsymbol{b}}_i) \\
\dot{\hat{\boldsymbol{b}}}_i &= \boldsymbol{f}_{dyn,ball}(\hat{\boldsymbol{b}}_i; \hat{\boldsymbol{w}}, \boldsymbol{g}) + \boldsymbol{K}_{vel}(\boldsymbol{y}_i - \hat{\boldsymbol{b}}_i) \\
\dot{\hat{\boldsymbol{w}}} &= \boldsymbol{K}_{\text{wind}} \frac{1}{N_{\text{balls}}} \sum_{i=1}^{N_{\text{balls}}} (\boldsymbol{y}_i - \hat{\boldsymbol{b}}_i)
\end{aligned} \tag{3.14}$$

The estimate for wind speed is adapted using a Luenberger observer with a constant air velocity model. As shown in eq. (3.15), the global estimate $\hat{\boldsymbol{w}}$ can be expressed as the average of an individual estimate per ball.

$$\begin{aligned}
\hat{\boldsymbol{w}}(t) &= \int_0^t \boldsymbol{K}_{wind} \frac{1}{N_{\text{balls}}} \sum_{i=1}^{N_{\text{balls}}} (\boldsymbol{y}_i(t) - \hat{\boldsymbol{b}}_i(t))dt \\
&= \frac{1}{N_{\text{balls}}} \int_0^t \sum_{i=1}^{N_{\text{balls}}} \boldsymbol{K}_{wind}(\boldsymbol{y}_i(t) - \hat{\boldsymbol{b}}_i(t))dt \\
&= \frac{1}{N_{\text{balls}}} \sum_{i=1}^{N_{\text{balls}}} \int_0^t \boldsymbol{K}_{wind}(\boldsymbol{y}_i(t) - \hat{\boldsymbol{b}}_i(t))dt \\
&= \frac{1}{N_{\text{balls}}} \sum_{i=1}^{N_{\text{balls}}} \int_0^t \dot{\hat{\boldsymbol{w}}}_i(t)dt \\
&= \frac{1}{N_{\text{balls}}} \sum_{i=1}^{N_{\text{balls}}} \hat{\boldsymbol{w}}_i(t)dt
\end{aligned} \tag{3.15}$$

Because of this equality, the system can be split into one estimator per flying ball as shown in eq. (3.16)

$$
\begin{aligned}
\dot{\hat{\boldsymbol{b}}}_{\boldsymbol{i}} &= \dot{\hat{\boldsymbol{b}}}_i + \boldsymbol{K}_{pos}(\boldsymbol{y}_i - \hat{\boldsymbol{b}}_i) \\
\dot{\dot{\hat{\boldsymbol{b}}}}_i &= \boldsymbol{f}_{dyn,ball}(\dot{\hat{\boldsymbol{b}}}_i; \hat{\boldsymbol{w}}, \boldsymbol{g}) + \boldsymbol{K}_{vel}(\boldsymbol{y}_i - \hat{\boldsymbol{b}}_i) \\
\dot{\hat{\boldsymbol{w}}}_i &= \boldsymbol{K}_{\text{wind}}(\boldsymbol{y}_i - \hat{\boldsymbol{b}}_i)
\end{aligned} \tag{3.16}
$$

The final estimate $\hat{\boldsymbol{w}}$ is the average of all ball-specific estimates. Since wind speed can only be measured when the ball is in free flight, balls that are currently in contact with the hand do not contribute to the average $\hat{\boldsymbol{w}}$. Upon leaving the hand, the ball can again provide data to estimate wind speed. Since the wind speed can change during the dwell time, the ball-specific estimate $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$ is initialized to the current global estimate $\hat{\boldsymbol{w}}$ to prevent jumps in the estimate.

4. Experiments

4.1. Experimental Setup

All experiments are performed with two simulated 4 DoF Barret WAM arms. A funnel-shaped end effector is attached to the end of each arm. The simulation is performed in a Mujuco environment with all contacts modeled as spring-damper systems. The arms are controlled by the inverse dynamics controller presented in section 3.5. For numerical optimization, IPOPT with the ma27 linear solver is used.

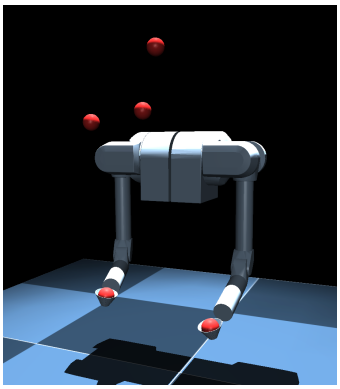


Figure 4.1.: Experimental Setup: 4 DoF Barret WAM arms with funnel shaped end effectors. In this picture, it juggles a 5-ball cascade.

The throwing position is offset by $0.2\,\mathrm{m}$ from the center plane. The default catching position is offset by $0.4\,\mathrm{m}$ from the center plane. The balls were simulated with mass $m_{\mathrm{ball}} = 0.067\,\mathrm{kg}$ and radius $r_{\mathrm{ball}} = 0.0375\,\mathrm{m}$.

Air drag is simulated using the eq. (2.2) with air density $\rho_{\text{air}} = 1.293\,\text{kg}\,\text{m}^{-3}$ and constant drag coefficient of the ball $c_d = 0.47$. Although drag coefficients technically vary with the Reynolds number, this is rarely simulated since the drag coefficient c_d is generally nearly constant in a wide range of Reynolds numbers. For a smooth sphere, the drag coefficient is around 0.47 in the range of Reynolds numbers $\text{Re} \in [10^3, 10^5]$ [12]. We use this approximation since we measure Reynolds numbers between 5000 and 25000 for our balls in the simulation.

All system components described in chapter 3 are parameterized by an array of values. Parameters that are influential for juggling success will be discussed in the analysis of the results. A broad set of parameter combinations is tested to find the specific settings at which specific juggling patterns can be juggled.

Each combination of hyperparameters is evaluated on the simulated system. In the deterministic case, one trial per combination is performed. For stochastic environments, 50 trials per combination are executed to enable grounded statistical statements. Trials terminate once a ball is dropped or the maximum number of 100 catches is reached.

4.2. Metrics and Comparision

We evaluate the following three metrics to quantify the performance of the developed algorithm.

1. *Number of Catches*: It has to be ensured that the patterns are juggled successfully.
2. *Solver Run Time*: The runtime is measured to differentiate between two equally successful hyperparameter configurations. This is especially interesting from the viewpoint of applicability to the actual system since the algorithm has to run in real time.
3. *Estimation Error*: To measure the quality of the proposed estimate, the estimation error $E_{\text{estimation}} = \frac{1}{N_{\text{steps}}} \sum_{k=1}^{N_{\text{steps}}} \|\hat{\boldsymbol{w}}_k - \boldsymbol{w}_k\|$ is measured.

Because of the huge dimensionality of the parameter space for the algorithm, not all parameters can be shown in the following figures. Unless stated otherwise, the maximum number of catches and the minimum planning time are shown in the figures. For example: Suppose the relation between siteswap patterns, constraints setting, and catch number is plotted, and one pattern with one constraint can be juggled at different cycle durations.

In that case, the maximum of the juggled catch numbers over the different cycle times is shown.

4.3. Siteswaps with and without Air Drag

Questions:

1. What constraints are needed for different site swap patterns?
2. How do the cycle duration and controller gains affect the physical jugglablity of the pattern?
3. Does the addition of air drag affect the juggling performance?

To test the capability of the proposed system, the following selection of siteswaps 3, 5, 7, 744, 672, 85525, and 9 were juggled with and without simulating air drag. The set of test sequences does not include 1 cascade since it corresponds to a handover of one ball without flight duration. The problem of handing over balls between unactuated end effectors without throwing the balls was outside the scope of this work. Between trials, constraint setting, cycle duration and inverse dynamics controller gains $\boldsymbol{K}_p$ are varied. The number of throws is recorded. The effects of air drag on all three variables are assessed. The influence on the *number of catches* can be seen in fig. 4.2.

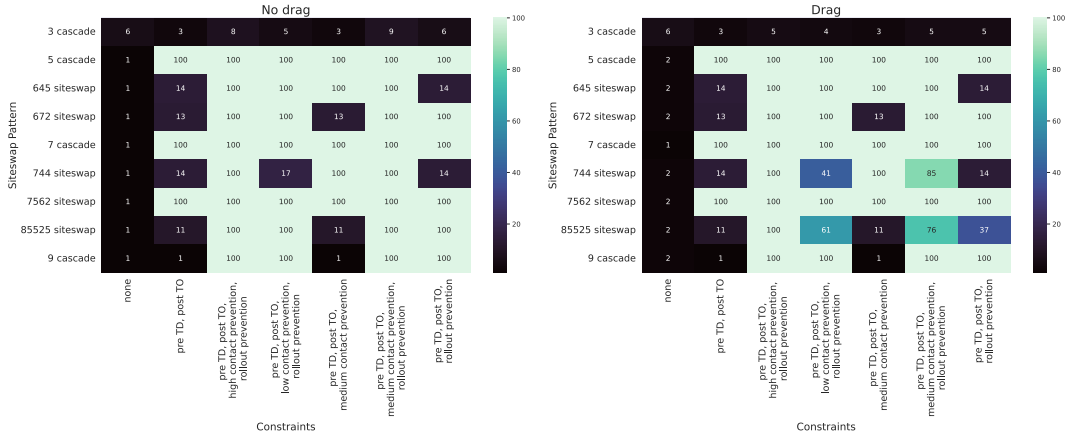


Figure 4.2.: A heat map showing the number of catches of juggling attempts of different patterns with different constraints. The number of catches is limited to 100.

Uniform patterns like the 5 and 7 ball cascades and the 7562 siteswap can be juggled successfully with the *pre touch down* and *post take off* constraint Ploeger et al. proposed in [15]. Additional constraints are needed for most siteswap patterns and the high 9 uniform pattern. This shows that the proposed constraints are essential for siteswap juggling in general.

The results also show that different site swap patterns require different constraints. For siteswaps including lower throws (e.g., 744) to the same hand, the *pre touchdown contact prevention* constraint is needed. Siteswaps with a 2 throw followed by a higher throw (e.g., 672) can only be juggled when applying the *rollout prevention*.

The 3 cascade cannot be juggled with any combination of constraints, cycle time, and controller gain we evaluate here. The reason for this might be the low angle of the flight trajectory at touchdown, leading the ball to bounce out of the funnel.

The effects of cycle time on the number of catches can be seen in fig. 4.3.

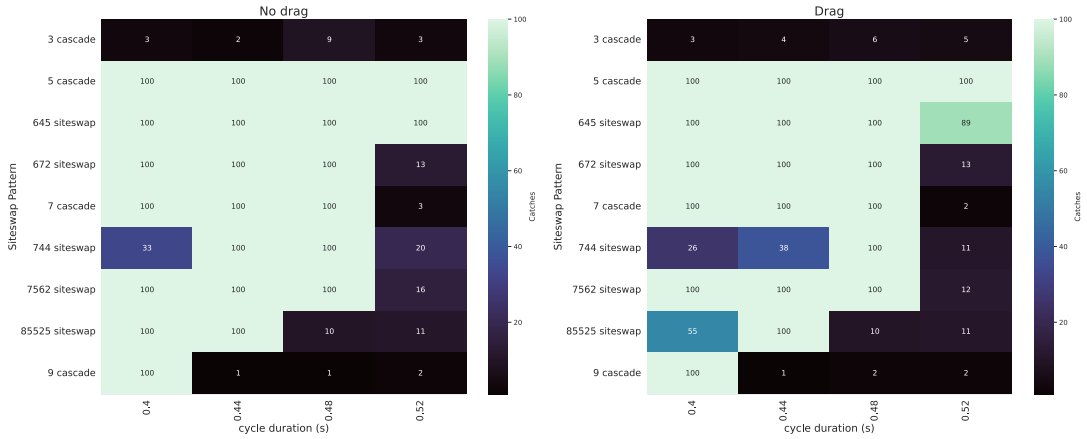


Figure 4.3.: A heat map showing the number of catches of juggling attempts of different patterns with different cycle times. The number of catches is limited to 100.

For most patterns, the juggling performance is better at lower cycle time. Higher cycle times mean longer flight times, increasing takeoff velocity and making the system less stable.

The influence of the proportional controller gain $\boldsymbol{K}_p$ on the *number of catches* can be seen in fig. 4.4.

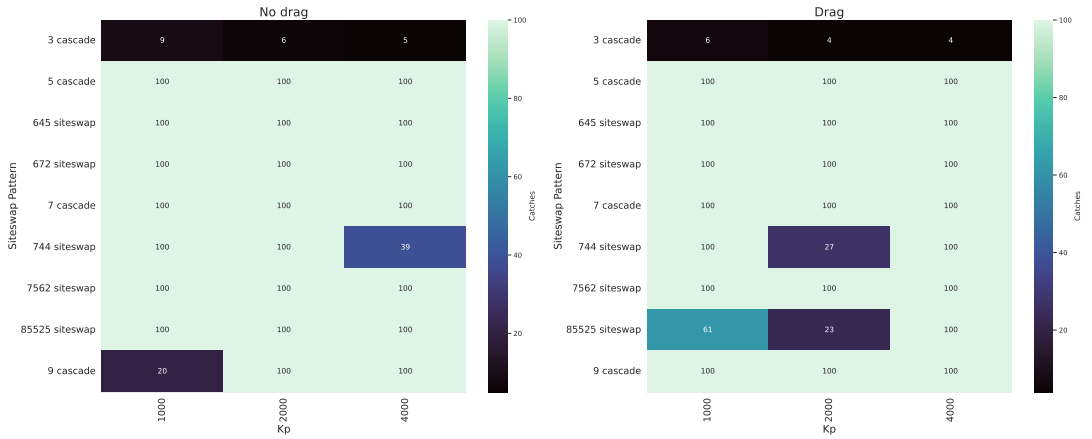


Figure 4.4.: A heat map showing the number of catches of juggling attempts of different patterns with $\boldsymbol{K}_p$ controller gains. The number of catches is limited to 100.

.

For most siteswap patterns, the controller gain does not influence juggling performance. Further investigation has to be conducted why the 744, 85525, and 9 pattern are sensitive to the controller gain and how air drag influences this sensitivity.

In general, it can be stated that air drag has little influence on juggling performance, even though we use relatively light balls even for juggling standards.

4.4. Juggling with Static Wind

Questions:

1. What is the maximal wind speed the system can juggle?
2. How does the system's knowledge of wind speed affect this limit?

To evaluate the resistance of our algorithm to wind under idealized conditions, we measured the maximum wind speeds in X (back to front), Y (left to right), and Z (below to above) direction at which the system can juggle 5 ball with $T_{\text{cycle}} = 0.4\,\text{s}$ and constraints *pre touchdown*, *post takeoff*, *pre touchdown contact prevetion* and *rollout prevention*. Successfully juggling at a given wind speed is defined as reaching 100 catches. Positive and negative integer wind speeds along each axis were tested until failure. We compared the performance of a system with perfect knowledge of the wind speed and one which assumes the wind to be zero at all times. The results are visualized in Figure 4.5.

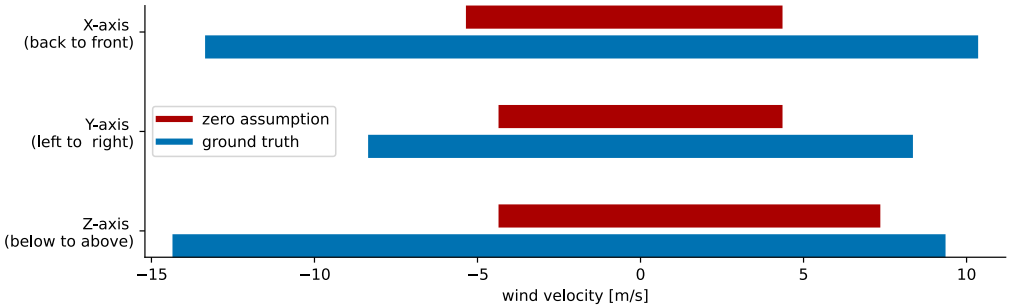


Figure 4.5.: Wind speed intervals along each axis in which successful juggling (100 catches or more) is possible. The intervals in which an algorithm without knowledge of the wind speed can juggle are shown in red. The adaptive algorithm with ground truth knowledge about the wind speed can juggle at windspeeds in the blue intervals.

Figure 4.5 shows that the system can adapt to low wind speeds without any knowledge of the wind by just adapting the catching position for incoming balls. Without knowledge about the wind, the system can cope better with frontal wind (negative X) than with

backward wind (positive wind) since the latter pushes the balls away from the robot's torso and out of the arms' workspace. With adaptation to wind, the system is equally resistant to frontal and rear wind. The interval is improved from $[-5, 3]$ to $[-8, 8]$. Since the environment is symmetric along the XZ plane, the Y intervals are also symmetric around 0. By adapting takeoff velocity as can be seen in fig. 3.9, the wind speed interval can be extended from $[-4, 4]$ to [-8,8]. The system can adapt to wind from above (negative Z) by throwing the balls upwards faster. Wind adaptation can extend the Z- interval from $[-4, 7]$ to $[-14, 9]$.

It is to be observed that this experiment only determines theoretical upper bounds. Sudden changes in wind speed and correlated effects of wind vectors that do not exclusively act in the direction of one of the cartesian axis are ignored in this context.

4.5. Juggling with Non-stationary Wind

In this experiment, the wind speed is sampled from a Wiener process. The Wiener process $\boldsymbol{W}_t, t \in \mathbb{R}$ is a time-continuous stochastic process. Its key property is that $W_s - W_d \sim \mathcal{N}(\mathbf{0}, (s-d)\boldsymbol{\Sigma})$: Given the value of W_d, W_s is normal distributed around W_d with Variance $\boldsymbol{(s-d)\Sigma}$. One sample trajectory of wind speed in all three axes with boundary condition $\boldsymbol{w}(0) = \mathbf{0}$ can be seen in fig. 4.6.

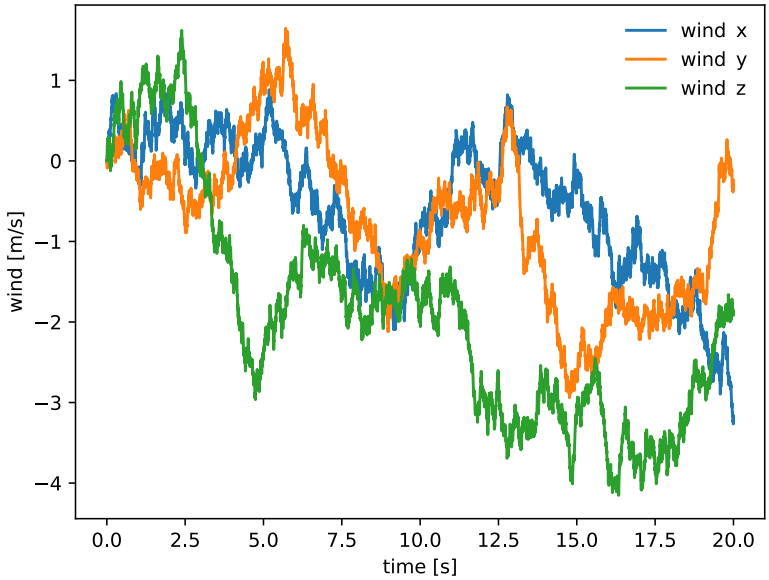


Figure 4.6.: Figure of wind speed over time sampled from a Wiener process with $\Sigma = 0.75\boldsymbol{I}$.

Questions:

1. How accurate is the observer with different estimation gains?
2. Does wind speed estimation and adaptive planning improve the system's performance?
3. Does the direction of the wind affect the number of catches?

In this experiment, only the uniform 5 ball cascade is analyzed. We examine the observer system (eq. (3.16)) with estimation gains $\boldsymbol{K}_{\text{wind}} = 100000\boldsymbol{I}, 500000\boldsymbol{I}, 700000\boldsymbol{I}$ with unchanged gains $\boldsymbol{K}_{\text{pos}} = 1000\boldsymbol{I}$ and $\boldsymbol{K}_{\text{vel}} = 1000\boldsymbol{I}$. One set of trials is performed with perfect knowledge of the wind speed (*groundtruth*). Wiener wind with $\boldsymbol{\Sigma} = 0.25\boldsymbol{I}, 0.5\boldsymbol{I}, 0.75\boldsymbol{I}, 1\boldsymbol{I}$ are tested. Over different experiments, the wind is applied on the following subspaces: X, Y, Z, XY, and XYZ.

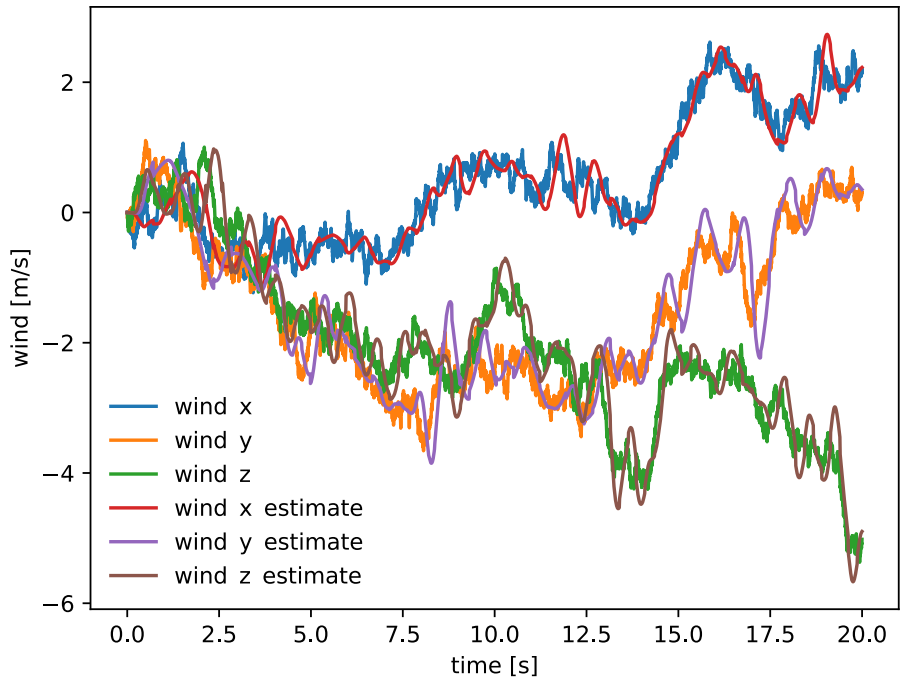


Figure 4.7.: Figure of wind speed over time sampled from a Wiener process with $\boldsymbol{\Sigma} = 0.75\boldsymbol{I}$ and the corresponding estimated by an observer with $\boldsymbol{K}_{\text{wind}} = 500000\boldsymbol{I}$.

Comparision of Different Estimator Gains

We first assessed the estimation accuracy of the observer with the different estimation gains $\boldsymbol{K}_{\text{wind}}$. The results are presented in fig. 4.8. An example of wind estimation over time can be seen in fig. 4.7. Since the observer with $\boldsymbol{K}_{\text{wind}} = \mathbf{0}$ always estimates the wind as $\mathbf{0}$, the estimation error reflects the stochastic nature of the Wiener process. For observers with $\boldsymbol{K}_{\text{wind}} \neq \mathbf{0}$, all average estimation errors range between $0\,\mathrm{m\,s^{-1}}$ and $1\,\mathrm{m\,s^{-1}}$. Generally $\boldsymbol{K}_{\text{wind}} = 100,000\boldsymbol{I}$ has a higher estimation error than $\boldsymbol{K}_{\text{wind}} = 500,000\boldsymbol{I}$ and $\boldsymbol{K}_{\text{wind}} = 700,000\boldsymbol{I}$. The estimation error generally increases with the increase in Σ since the environment becomes more stochastic.

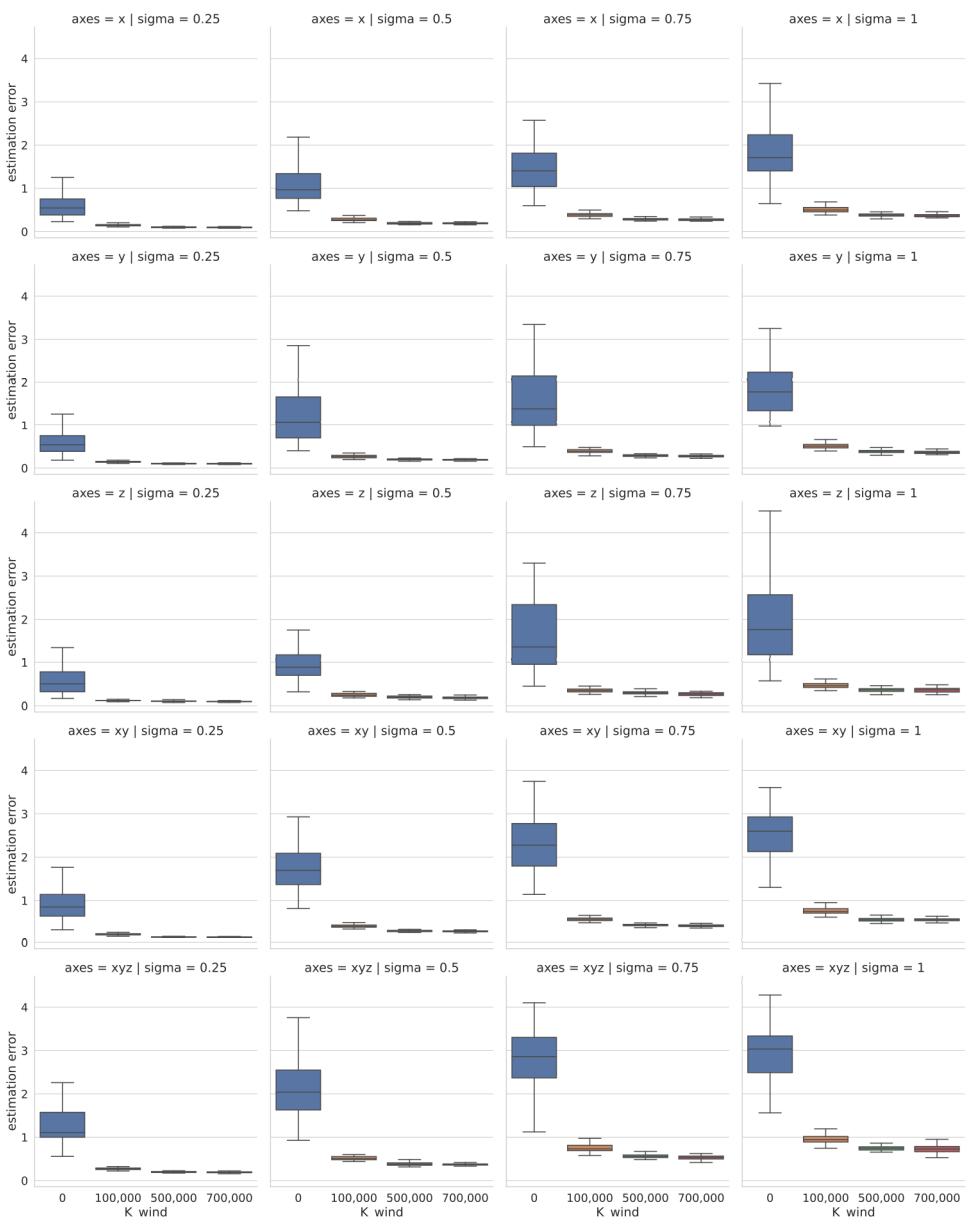


Figure 4.8.: This figure show the *estimation error* for different observer gains $\boldsymbol{K}_{\text{wind}}$ at different winds. Each row shows winds applied in one subspace. Each column corresponds to a different value for the variance $\boldsymbol{\Sigma}$ of the Wiener wind. Different estimation gains $\boldsymbol{K}_{\text{wind}}$ are shown in different colors.

Effect of Wind Knowledge on Juggling Performance

For the following comparison for the number of catches, we choose the observer with $\boldsymbol{K}_{\text{wind}} = 500000\boldsymbol{I}$, although it has a slightly higher estimation error than $\boldsymbol{K}_{\text{wind}} = 700000\boldsymbol{I}$, to prevent oscillations in the estimate. It is tested alongside a system with perfect knowledge of the wind speed (*groundtruth*) and a system that assumes zero wind speed ($\boldsymbol{K}_{\text{wind}} = \boldsymbol{0}$)

The results can be seen in fig. 4.9. The maximum number of catches is usually reached on the X- and Y-axis, although the system with zero assumption has a higher variance than proper estimators. A high variance can be observed on the Z-axis with all estimator settings. This implies that the system generally is not resistant to unexpected changes in the wind velocities in Z- direction. This hypothesis is further supported when comparing Wiener wind on all axis with Wiener wind only in the XY direction. In the XY direction, the planning generally works reliably, producing many catches, while it performs poorly with Wiener motion on all axis.

Poor performance with wind along the Z axis can be explained by analyzing the effect of wind on the touchdown time of the balls. Wind along the X and Y axis does not change touchdown time because it does not affect the vertical velocity of the ball, while wind along the Z axis accelerates or decelerates the ball vertically, altering the touchdown time. This is not a problem for known and constant wind because the optimization can compensate for the additional wind, However, random wind alters the time unpredictably. Since this disturbs the juggling rhythm, siteswap patterns are susceptible to alterations in flight duration. Consequently, random vertical wind is especially challenging.

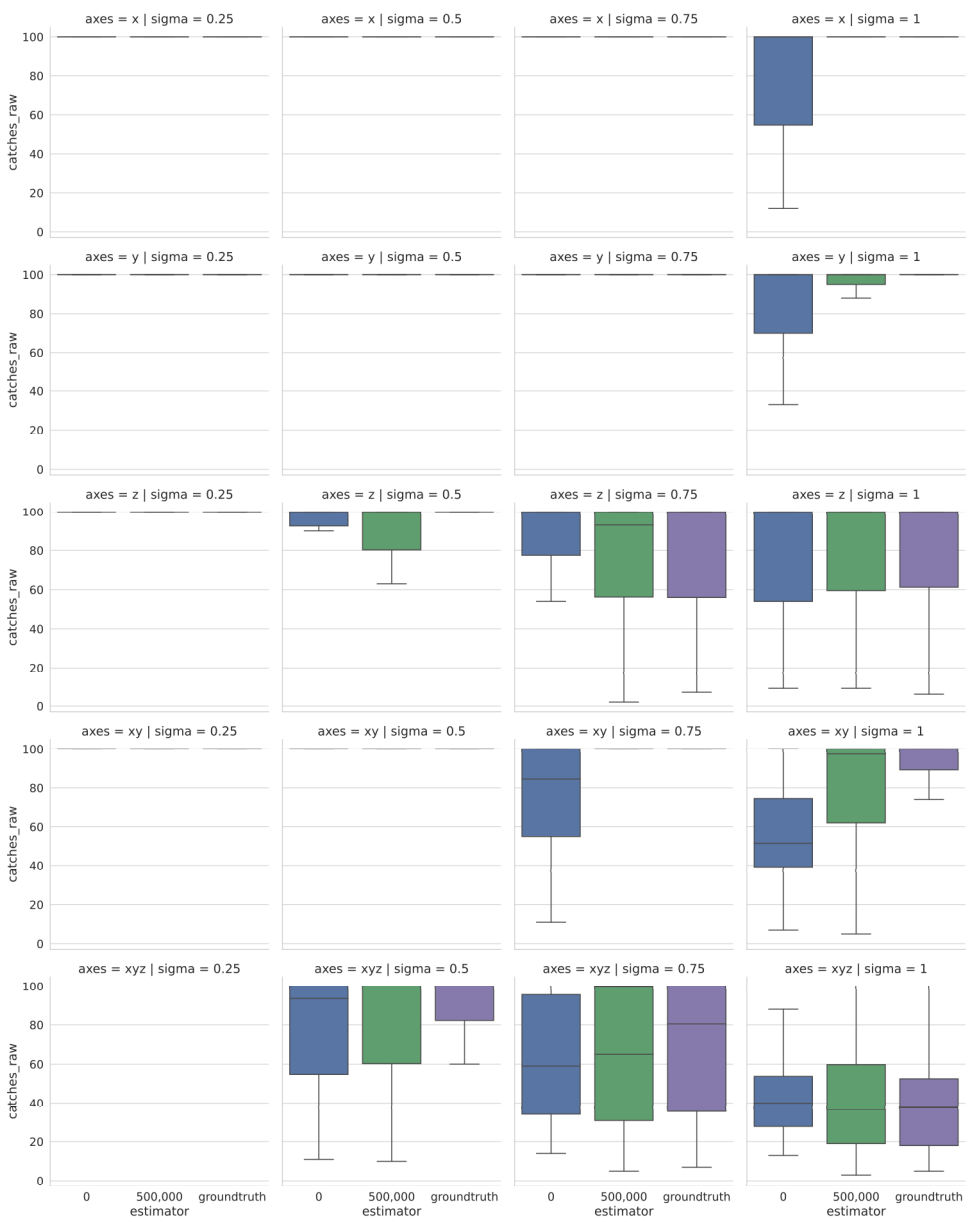


Figure 4.9.: This figure shows the *number of catches* for different observers at different winds. Each row shows winds applied along one axis. Each column corresponds to a different value for the variance Σ of the Wiener wind. Different observers are shown in different colors. Note that a thin line at $y = 100$ is a full box plot, meaning that all juggling attempts resulted in 100 catches.

Our experiments on adaptability to wind generated by a Wiener process demonstrate, that our adaptive throw velocity planning, in combination with our wind speed estimation, significantly improves resistance to horizontal (XY) wind. The experiments also show that vertical wind is challenging independent of the knowledge of windspeed at planning time.

4.6. Random Walk on Siteswap Graph

Questions:

1. Can our algorithm juggle random walks on the siteswap graph?
2. How does the set of allowed throws affect performance?
3. Is random walk juggling possible with air drag?

As described in section 2.2.1, all transitions in the siteswap graph correspond to a valid throw. This property can not only be used to transition from the ground state to a desired cyclic siteswap sequence, but also to generate a random sequence of juggle throws by transitioning randomly in each state. This results in a seemingly unordered pattern that still fulfills all assumptions of *semi-simple juggling*. Accordingly, it can be juggled.

We evaluated random walks at cycle durations $T_{\text{cycle}} = 0.44\,\text{s}, 0.48\,\text{s}$ with and without air drag. *Post takeoff*, *pre touchdown*, *pre touchdown contact prevention* and *rollout prevention* constraints were applied to the optimization. We also differentiated between different subsets of transitions $[0, 2, 4, 5, 6, 7]$ and $[0, 2, 4, 5, 6, 8]$ that were allowed to be included in the random walk. Note that 1 - throws and 3 - throws are not included in both sets because our system cannot execute either of them (see section 4.3) Each combination was executed 50 times to a maximum of 100 catches. The results are shown in fig. 4.10.

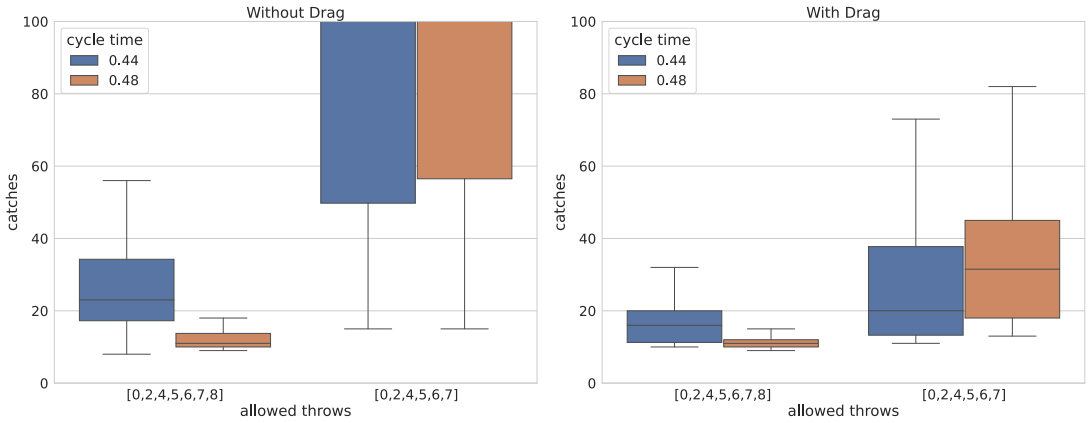


Figure 4.10.: Boxplot of *number of catches* archived with random siteswap graph walk with and without air drag and different cycle durations. Allowed throws are the set of throws that can be used to transition between the state in the siteswap graph. The *number of catches* is limited to 100.

In general random juggling with the $[0, 2, 4, 5, 6, 7]$ throw subset archives a median of 100 catches Random juggling with a bigger throw subset archives significantly lower catch numbers. A shorter cycle time benefits random juggling with $[0, 2, 4, 5, 6, 7, 8]$, while longer cycle times benefit random juggling with $[0, 2, 4, 5, 6, 7]$. Both random juggling variations perform considerably worth with air drag.

The siteswap graph can be used to generate a great variety of throw sequences that can be juggled on our system. Because of the big range of available throws, especially with the $[0, 2, 4, 5, 6, 7, 8]$ throw set, random graph walks are more difficult to juggle and more susceptible to air drag than regular juggling patterns.

4.7. Planning Times and Implications for Real-Time Applicability

Questions:

1. How do constraints and the cycle time affect joint space and throw velocity planning time?
2. Does the algorithm plan in real-time?
3. What constraints have the highest need and potential to be optimized?

Joint Space Planning Times

In fig. 4.11 planning times for joint space optimization (eq. (3.9)) in relation to the applied constraints are shown as heat maps.

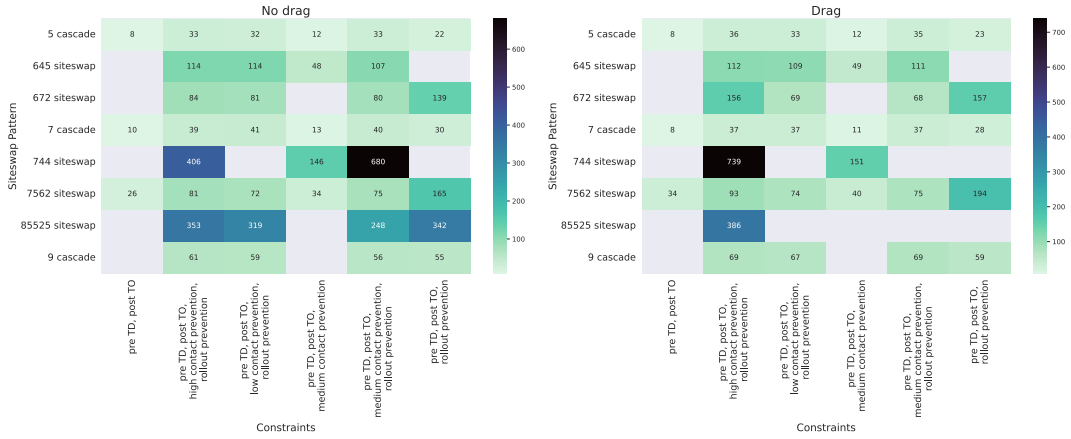


Figure 4.11.: A heat map showing the average joint space planning time $[\mathrm{ms}]$ of patterns with different constraint settings. Only combinations where the maximum catch number was reached are shown.

.

Only applying more constraints does not directly lead to an increase in planning time.

In patterns like the 5 or 7 cascades where the constraints are probably already fulfilled without actively enforcing them they lead to a lower increase in planning time when compared to patterns in which they have to be actively optimized, such as the 744 or the 85525 siteswap.

Figure 4.12 shows the relation of joint space optimisation time, cycle duration and applied constraints. Note that the mean of all successful attempts with 100 catches or more is shown. For cycle times $0.4\,\mathrm{s}$ to $0.48\,\mathrm{s}$ multiple patterns can be juggled while at $0.52\,\mathrm{s}$ at most the 5 ball cascade and the 645 siteswap can be juggled (see fig. 4.3). This decreases the mean since most constraints do not have to be actively enforced for the 5 pattern.

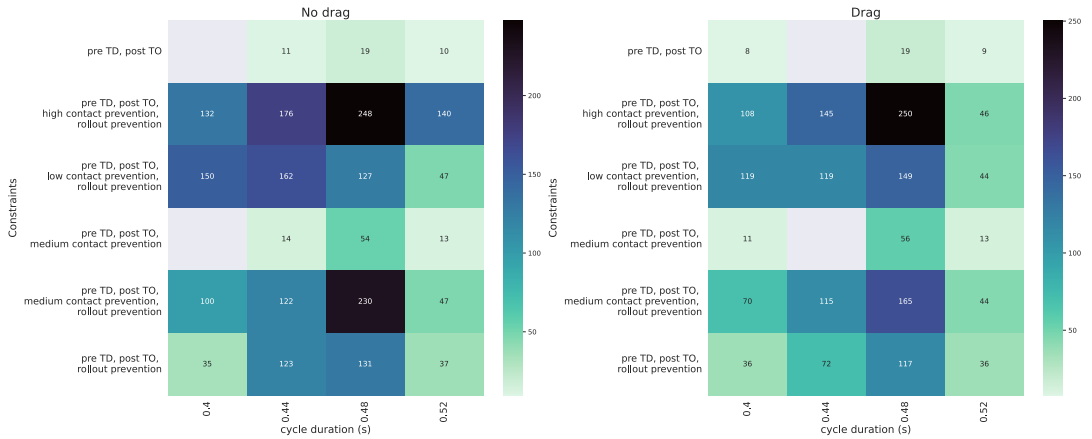


Figure 4.12.: This heatmap shows the mean of joint space planning times of all successfully jugglable sequences (100 catches) with respect to cycle duration and constraint set. Note that the mean of the joint space planning time for $T_{\mathrm{cycle}} = 0.52\,\mathrm{s}$ is almost entirely determined by the 5 ball cascade since only this sequence can be juggled at such a high cycle time.

In general joint space planning times increase with the number of applied constraints. The addition *pre touch down contact prevention* constraint only slighly increase the planning time while the *rollout prevention* constraint often increases it by more than 100%. This can be explained by the different complexities of the constraints. Both constraints include the forward kinematics, but the *pre touchdown contact prevention* constraint only includes an additional norm operation while the *rollout prevent* constraint includes a $\arcsin$ operation, which is highly non-convex.

For all constraint sets, longer cycle duration usually correspond to longer planning times since the optimization includes more time steps. An exception to this trend is $T_{\text{cycle}} = 0.52\,\text{s}$ as previously explained. Joint space planning times are roughly comparable with and without air drag.

Throw Velocity Planning Times

In fig. 4.13, the planning time for throw velocity optimization eq. (3.12) are shown in heatmap. Planning times without drag are generally lower than planning times with drag.

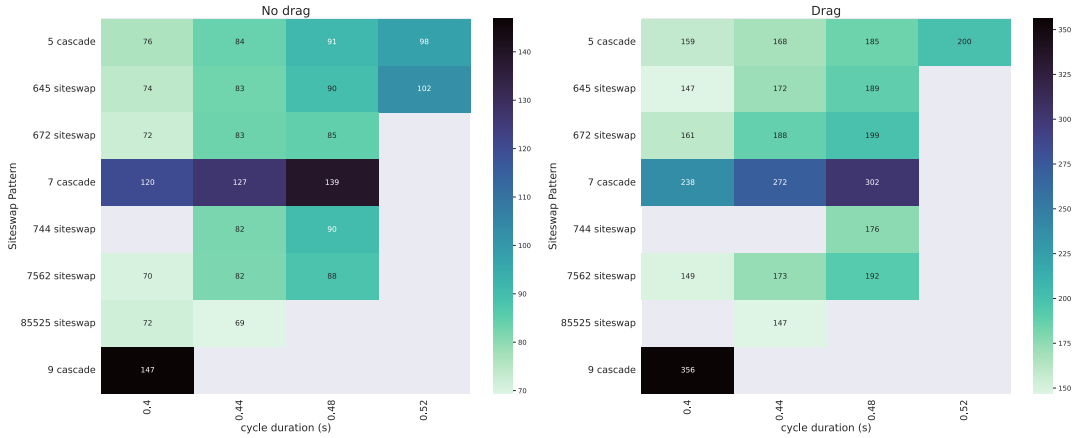


Figure 4.13.: A heat map showing the average throw velocity planning time $[\text{ms}]$ of patterns with different cycle times. Only combinations in which the maximum catch number is reached are shown.

.

The planning time increases with the cycle duration since optimization has to be performed over more time steps. Note that the comparison between drag and no drag in relation to constraint sets and throw planning time is omitted here since the throw velocity planning time is independent of the hand's task space constraints.

Real Time Applicability

For throws to be planned in real-time, two hand movements have to be planned. Planning includes the optimization of the takeoff velocity as well as optimization in joint space. Assuming a cycle duration of $0.44\,\mathrm{s}$ and flight durations that are long enough to allow for accurate trajectory estimation of a ball before its catch has to be planned, the execution time for planning is bound by $220\,\mathrm{ms}$ because two catches have to be planned per cycle duration. In most scenarios, our joint space trajectory planner already fulfills this requirement. There are two main approaches to speed up optimization in the rest of the scenarios.

Firstly, previous optimization solutions can be cached and used as an initial value for consecutive optimizations. Ploeger et al. [15] already employed this caching approach for optimization in joint space. This system is also used here but has a major drawback for siteswap juggling. Since only the previous result is cached, but the throw heights vary from throw to throw, expensive optimization has to be performed constantly to find a solution outside the proximity of the previous one. Furthermore, we currently do not cache the results of throw planning. Both optimizations will be augmented with improved caching approaches, which store more solutions in an ordered manner to initialize the solvers optimally.

Secondly, the constraints themselves could be simplified by precomputing inverse kinematics to find forbidden areas in joint space corresponding to the constraints in task space. Finding a different formulation of the *roll prevention* constraint without a trigonometric function would also greatly improve performance.

5. Discussion

This work presents an algorithm for planning joint space trajectories to juggle siteswap patterns. It utilizes an implementation of a siteswap graph to generate a throw sequence that has to be executed for a desired pattern. To make siteswap juggling possible, we identify two previously unreported trajectory constraints for robotic siteswap juggling and integrate them into our planning algorithm. We show that both the *pre touch down contact prevention* constraint and the *rollout prevention* constraint are essential for juggling with open hands. We implement a siteswap graph and demonstrate the juggability of various siteswap patterns on our 4 DoF robots in simulation.

Additionally, we employ a Luenberger observer to estimate wind speed from ball positions and present an optimization formulation to use this information to find the optimal takeoff velocity. Our experiments show the robustness of our adaptive algorithm, especially when compared to the nonadaptive predecessor without throw velocity optimization. Our algorithm can juggle at wind speeds of up to $15\,\mathrm{m\,s^{-1}}$ and handle wind fluctuations from a Wiener Process with up to $\Sigma = 0.75\boldsymbol{I}\mathrm{m^2\,s^{-2}}$.

Our algorithm can plan random walks on the siteswap graph, which can be successfully executed in simulation. Drag leads to a big decrease in performance on random graph walks but not on regular siteswap, indicating that random graph walks requires further improvement of the algorithm. The wide range of different throws in a random pattern might not all be optimizable with one global set of constraints but demand individual constraints for each throw.

Interestingly, our algorithm can not juggle a 3-ball cascade, one of the simplest juggling patterns. This could be explained by the low flight time, leading to a low touchdown velocity angle. Additionally, the contact dynamics of the ball in the simulation are more challenging than the dynamics of juggling balls used to toss juggles on real robotic systems [14]. In the simulation, balls are modeled as simple spring-dampener systems, which makes them bounce easily. We will use sand-filled balls that do not bounce but are still

very rigid for experiments on the real robot. These balls are easier to manipulate while preserving throw accuracy.

Our algorithm is fast enough for real-time application in most scenarios. For more complex scenarios, runtime has to be optimized. Fortunately, several approaches to reduce runtime can easily be pursued.

6. Outlook

The central objective of future work is the speed up of the current controller algorithm. An obvious first step would be reimplementing the system in a low-level language. Another important approach would be the simplification and approximation of the presented constraints. Like some throwing and catching constraints, the proposed constraints could be translated from task into joint space to remove the need to perform forward kinematics during optimization. Forbidden subspaces within joint space corresponding to task space constraints could be pre-computed to improve runtime. A more advanced caching system for previous solutions that can be used to initialize the solvers for following optimizations would also greatly reduce computing time.

The experiments in section 4.6 show that juggling a sequence generated by a random walk on the siteswap graph is possible under ideal conditions but still has the potential to be improved. Different throws may require different constraint sets. Consequently, we aim to determine the best constraint settings for each throw height and possible dependencies between different consecutive throws and catches. We also want to investigate why uniform juggling with 3 balls does not work with our current approach.

In this work, we perform the hyperparameter search for the algorithm manually. Better hyperparameters could be found in a principled way using Bayesian optimization or episodic reinforcement learning. Episode-based reinforcement learning algorithms, such as the Cross-Entropy Method (CEM), could be applied to find better configurations within the parameter space.

Semi-simple juggling is only a small subgroup of robotic juggling and juggling in general. One direction that could be explored is robot-to-robot or human-to-robot partner juggling. In this scenario, the system would have to address similar challenges but with greater difficulty because it would have to adapt to its own mistakes, random disturbances, and another agent. Adjusting to the other agent's or human's rhythm and recovering from timing mistakes that lay beyond just a fraction of a cycle time is necessary.

Besides cooperative juggling, additional juggling types can be explored. Multiplexing - throwing, catching, or keeping multiple balls in hand at once - is a possible extension of the work. Similarly to this work, discrete tools such as graphs can model more complex juggling patterns. This topic would also require the design of a robotic hand that can be used for multiplex juggling since open-handed juggling, as described here, does not allow the controlled handling of multiple balls at once by one hand. However, this would entail its own set of simulation and control challenges.

Most importantly, we would like to employ our algorithm on a real robot as soon as possible!

Acknowledgements

I thank Kai Ploeger for his supportive yet demanding supervision, which motivated me to do my best. I thank Prof. Jan Peters for sparking my interest in robotics and enabling me to pursue this topic at the IAS. I thank my family and friends who inspire me.

Bibliography

[1] Paul Barsa et al. “Autonomous Juggling Robot”. In: *2019 IEEE MIT Undergraduate Research Technology Conference (URTC)*. Juggling on 2D Plane with marbels, also siteswaps. IEEE. 2019, pp. 1–4.

[2] Christopher Brueningsen et al. “Modeling air drag”. In: *The Physics Teacher* 32.7 (1994), pp. 439–441.

[3] Joe Buhler et al. “Juggling drops and descents”. In: *The American Mathematical Monthly* 101.6 (1994), pp. 507–519.

[4] M Buhler and Daniel E Koditschek. “From stable to chaotic juggling: Theory, simulation, and experiments”. In: *Proceedings., IEEE International Conference on Robotics and Automation*. IEEE. 1990, pp. 1976–1981.

[5] Jason Chemin and Jehee Lee. “A physics-based juggling simulation using reinforcement learning”. In: *Proceedings of the 11th ACM SIGGRAPH Conference on Motion, Interaction and Games*. 2018, pp. 1–7.

[6] Billy Gillen. “Remember the Force Hassan!” In: *Juggler’s World* 38.2 (1986), pp. 14–18.

[7] Russell Herman. “The Flight of Sports Balls”. In: ().

[8] Matthew Kelly. “An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation”. In: *SIAM Review* 59.4 (2017), pp. 849–904. eprint: `https://doi.org/10.1137/16M1062569`.

[9] Takahiro Kizaki and Akio Namiki. “Two ball juggling with high-speed hand-arm and high-speed vision system”. In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 1372–1377.

[10] Jens Kober, Matthew Glisson, and Michael Mistry. “Playing catch and juggling with a humanoid robot”. In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. IEEE. 2012, pp. 875–881.

[11] Arthur Lewbel. “History of Juggling”. In: *Boston College, March* (2002).

[12] NASA Glenn Research Center. *Beginner's Guide to Aeronautics: Drag of a Sphere*. https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/drag-of-a-sphere/. Accessed: July 5, 2023.

[13] Jorge Nocedal, Andreas Wächter, and Richard A Waltz. "Adaptive barrier update strategies for nonlinear interior methods". In: *SIAM Journal on Optimization* 19.4 (2009), pp. 1674–1693.

[14] Kai Ploeger, Michael Lutter, and Jan Peters. "High acceleration reinforcement learning for real-world juggling with binary rewards". In: *Conference on Robot Learning*. PMLR. 2021, pp. 642–653.

[15] Kai Ploeger and Jan Peters. "Controlling the cascade: Kinematic planning for n-ball toss juggling". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 1139–1144.

[16] Burkard Polster et al. *The mathematics of juggling*. Tech. rep. Springer, 2003.

[17] Fu Qifeng. "All our Yesterdays: Early Chinese jugglers quell war, entertain royalty". In: *Juggler's World* 39.1 (1987).

[18] Philipp Reist and Raffaello D'Andrea. "Design and analysis of a blind juggling robot". In: *IEEE Transactions on Robotics* 28.6 (2012), pp. 1228–1243.

[19] Marcia Riley and Christopher G Atkeson. "Robot catching: Towards engaging human-humanoid interaction". In: *Autonomous Robots* 12 (2002), pp. 119–128.

[20] Alfred A Rizzi and Daniel E Koditschek. "Further progress in robot juggling: The spatial two-juggle". In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. IEEE. 1993, pp. 919–924.

[21] Alfred A Rizzi and Daniel E Koditschek. "Progress in spatial robot juggling". In: (1992).

[22] Takeshi Sakaguchi, Yasuhiro Masutani, and Fumio Miyazaki. "A study on juggling tasks". In: *Proceedings IROS'91: IEEE/RSJ International Workshop on Intelligent Robots and Systems' 91*. IEEE. 1991, pp. 1418–1423.

[23] Takeshi Sakaguchi et al. "Motion planning and control for a robot performer". In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. IEEE. 1993, pp. 925–931.

[24] Stefan Schaal and Christopher G Atkeson. "Open loop stable control strategies for robot juggling". In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. IEEE. 1993, pp. 913–918.

[25] Stefan Schaal, Christopher G Atkeson, and Dagmar Sternad. “One-handed juggling: A dynamical approach to a rhythmic movement task”. In: *Journal of motor behavior* 28.2 (1996), pp. 165–183.

[26] Claude E Shannon. *Scientific aspects of juggling*. 1993.

[27] Vahram Stepanyan and Kalmanje S Krishnakumar. “Estimation, navigation and control of multi-rotor drones in an urban wind field”. In: *AIAA information systems-AIAA infotech@ aerospace*. 2017, p. 0670.

[28] Lieven Vandenberghe. “EE236A - Linear Programming”. In: 2013.

[29] Andreas Wächter and Lorenz T Biegler. “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. In: *Mathematical programming* 106 (2006), pp. 25–57.

[30] Margaret Wright. “The interior-point revolution in optimization: history, recent developments, and lasting consequences”. In: *Bulletin of the American mathematical society* 42.1 (2005), pp. 39–56.

A. Appendix

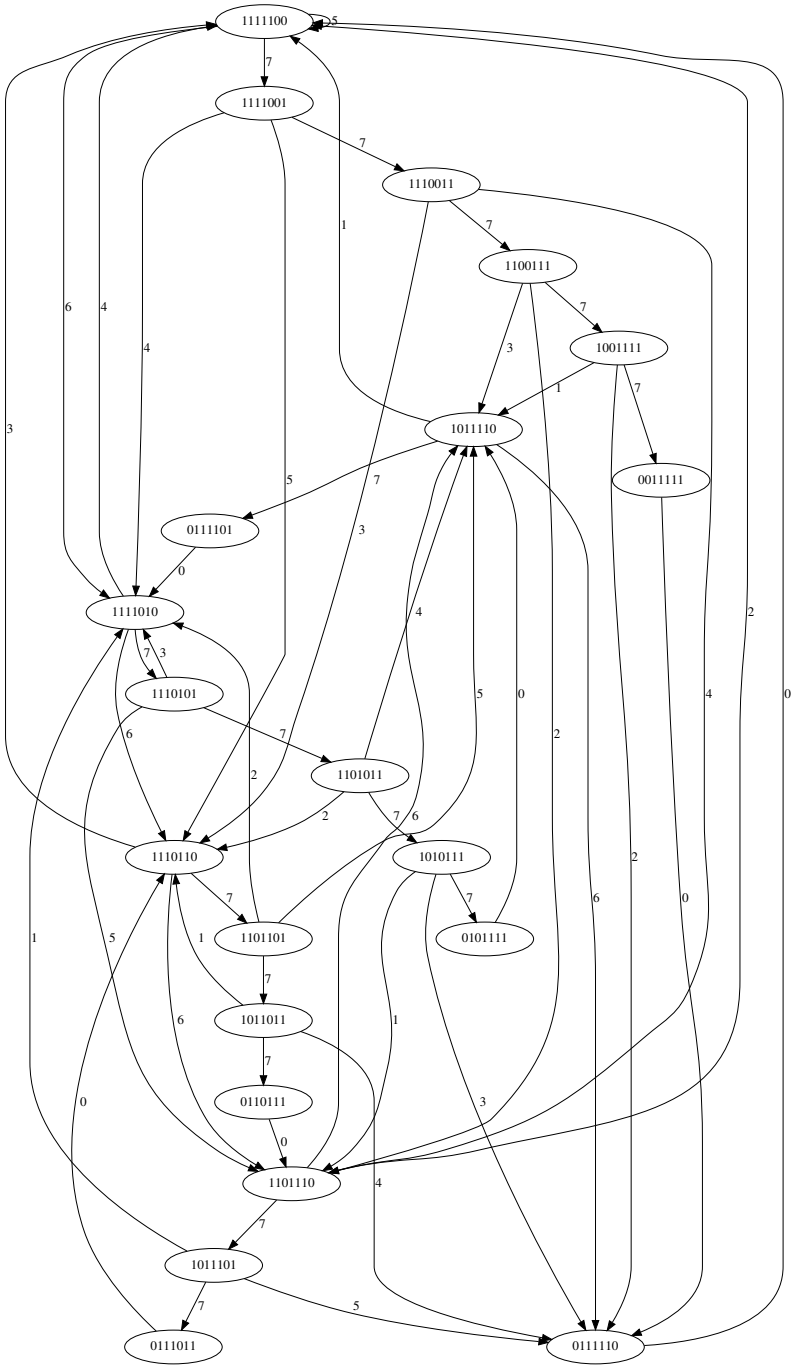


Figure A.1.: Figure of a the full siteswap graph for $N_{\mathrm{balls}} = 5$ and $h_{\max} = 7$