Archetecture for High Degree of Freedom Complex Control Systems

A Thesis

Submitted to the Faculty

of

Drexel University

by

Daniel Marc Lofaro

in partial fulfillment of the

requirements for the degree

of

Doctor of Philosophy in Electrical and Computer Engineering Engineering

June 2013

Table of Contents

List of Tables

List of Figures

0.1 Introduction

0.1.1 What I am doing and why

- Creating a system that can monitor its self
- Avoid self collisions
- Plan future movements
- Proform these plans with a higher archical structure such that it stays stable and safe among all else.

0.1.2 Types of Robots

0.1.3 Platform

The Hubo2 Plus series robot is a 130 cm (4' 3") tall, 42 kg (93 lb) full-size humanoid commonly refereed to as Hubo. The Hubo series was designed and constructed by Prof Jun-Ho Oh at the Hubo Lab in the Korean Advanced Institute of Science and Technology (KAIST) in Daejeon, South Korea [**?**]. Hubo has 2 arms, 2 legs and a head making it anthropomorphic to a human. It contains 6 degrees of freedom (DOF) in each leg, 6 in each arm, 5 in each hand, 3 in the neck, and 1 in the waist; all totaling 38 DOF. All joints of the major joints are high gain PID position controlled with the exception of the fingers. The fingers are open-loop PWM controlled. The sensing capability consists of a three axis force-torque (FT) sensor on each leg between the end of the ankle and the foot as well as between the arm where it connects to the hand. Additionally it has an inertial measurement unit (IMU) at the center of mass and accelerometers on each foot. The reference commands for all of the joints are sent from the primary control computer (x86) to the individual motor controllers via two Controller Area Network (CAN) buses. This is the same communications bus

found in most modern motor vehicles. There are currently eight Hubo's functioning in the United States as of December 2012. Four reside at Drexel University and one at Georgia Tech, Purdue, Ohio State and MIT. Jaemi Hubo is the oldest of the Hubos in America and has been at the Drexel Autonomous Systems Lab[1] (DASL) since 2008 [**?**]. Fig. 1 shows the major dimensions of Hubo.

A full-scale safe testing environment designed for experiments with Jaemi Hubo was created using DASL's Systems Integrated Sensor Test Rig (SISTR) [1]. Additionally all algorithms are able to be tested on miniature and virtual versions of Jaemi Hubo prior to testing on the full-size humanoid through the creation of a surrogate testing platform for humanoids [2].

0.1.4 Background

Kinematic Planning

Kinematic planning focuses on creating and testing valid trajectories for series kinematic manipulators. The focus of this research is on high degree of freedom (DOF), high-gain, position controlled mechanisms. The works are chosen as it pertains to end-effector velocity control. Throwing and hitting are examples of end-effector velocity control. The goal is to have the end-effector moving at a specific rate in a specific direction. In most cases it demands whole-body coordination to achieve a desired end-effector velocity. Whole-body coordination is different for planted robots and un-planted robots.

Planted robots are robots where the base is attached to the ground or the base is significantly more massive then the manipulator. Planted robots do not have to worry about balance consternates.

Un-planted robots are robots that have an manipulator that is not significantly ligher

[1]Drexel Autonomous Systems Lab: http://dasl.mem.drexel.edu/

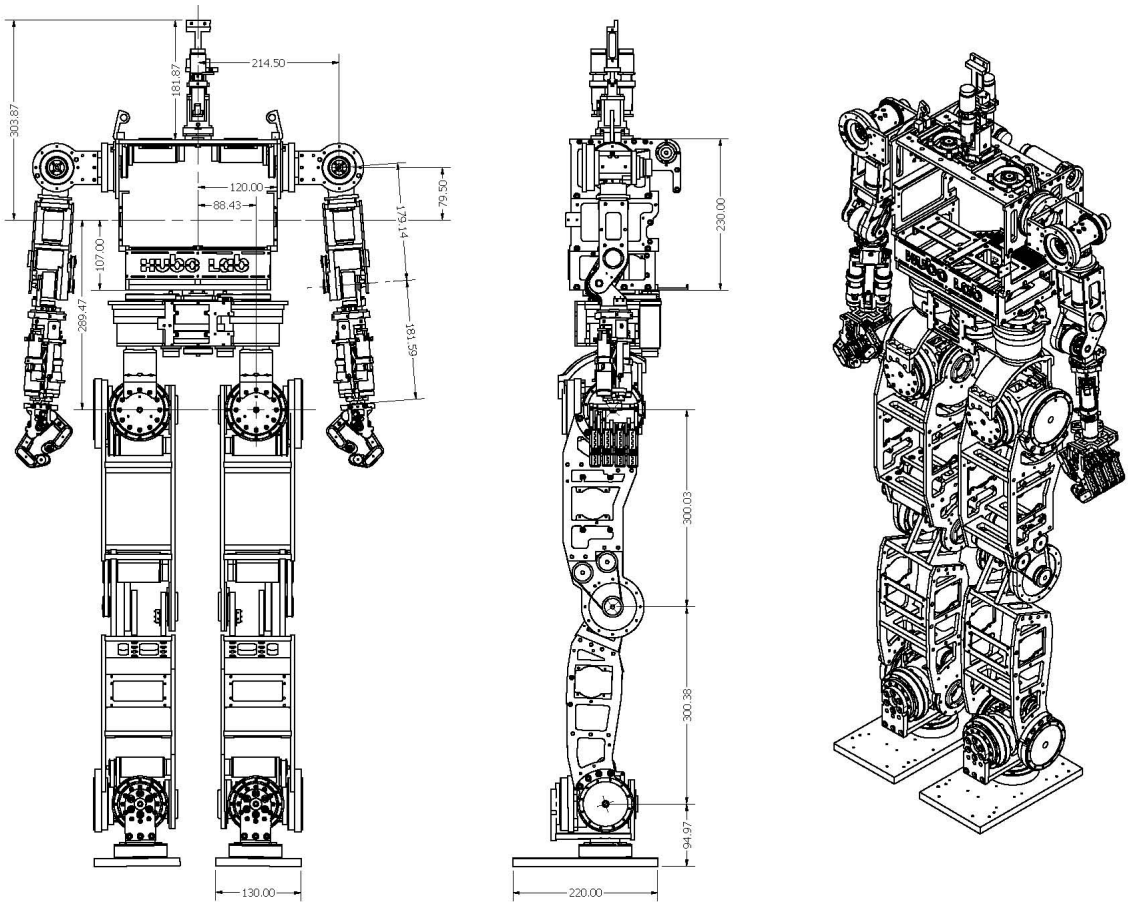


Figure 1: Hubo2 Plus platform: 38 DOF, 130 *cm* tall full-size humanoid robot weighing 37 *kg*.

then the base. In addition the robot is not physically attached to the ground. This results in the robot needing to satisfy balance constraints. In the static case if the robot satisfies the zero moment point (ZMP) criteria it will remain stable [3]. When the manipulator moves quickly, as in the case of pitching or throwing, such upper-body motions if not coordinated with the lower-body, can cause the humanoid to lose balance.

End-Effector Velocity Control

End-effector velocity control (EEVC) is the act of moving your manipulator at a given speed through space at a given velocity. EEVC is being looked at as the mass of the end-effector does not change. Thus by controlling the velocity we also control the inertia. In addition I will be exploring EEVC as it pertains to manipulating objects. Through my research I have found that end-effector velocity control can be broken up into four major categories: *Time and location sensitive*, *location sensitive*, *time sensitive*, and *time and location insensitive*.

Time and Location Sensitive: If your velocity control is time and location sensitive it means that your end effector needs to have a given velocity at a specific time in a specific location or the task fales. Hitting a baseball with a bat is an example of *time and location sensitive* EEVC. If the bat has the correct velocity but not at the correct time it will not hit the ball or the ball will not go in the desired place. The same goes for if it does not have the correct location but does have the correct velocity. It is important to note that the manipulator only has instantaneous control over the object at the instant of contact. Other examples include playing the piano, hitting a tennis ball with a racquet, a moving soccer ball with a foot or any other task that requires to *hit* a *moving* object.

Location Sensitive: If your velocity control is location sensitive it means that it only matters that the velocity occurs at a given location. The time it takes to reach that velocity will not effect the results. Hitting a nail with a hammer is a prime example of location sensitive EEVC. The nail is not moving but it does need to be hit in a given location with a given velocity. The vector of the velocity is determined by the required angle the nail needs to be hit at. In this example the nail is not time dependent and can be hit any time. Hitting it a $t = N$ or $t = N + 1$ will not effect the results. It is important to note that the manipulator only has instantaneous control over the object at the instant of contact. Other examples of location sensitive end-effector velocity control are hitting a golf ball with a club, hitting a pool ball with the cue, and other activities that require a given location and direction of manipulation but are not time dependent.

Time Sensitive: If the location were the end-effector achieves a given velocity is not required to complete the task but the time when it happens is required it is considered *time sensitive* EEVC. This means that the end-effector can move in any region it desired as long as the end effector achieves a given velocity at a given time. The end-effecter's velocity can be dependent on the location achieved but the location is an independent variable and the velocity is the dependent variable. It is important to note that the manipulator control over the object during the entirety of the motion. This typically means that the manipulator is holding the object until the release stage. An example of this is throwing a baseball to first base to get someone out. Throwing the ball side arm, over arm, or even underarm does not matter as long at it is released at the correct time with the correct velocity to get it ball to the first-baseman to get the runner out. Other example of time sensitive EEVC are any

other instance where an object is thrown within a given time.

Time and Location Insensitive: If the location and the time of when the end-effector achieves a given velocity does not matter it is considered time and location insensitive. The end-effecter's velocity can be dependent on the location achieved but the location is an independent variable and the velocity is the dependent variable. In this case the manipulator has control over the object until the release stage. Examples of this would be pitching a baseball, bowling, throwing a grenade or horseshoes etc. Throwing is an example of when the end-effector's velocity holds a higher priority over the position.

Mechanisms with only a single degree of freedom are restricted to throwing in a plane. 2-DOF mechanisms are able to throw in R^3 space with the correct kinematic structure. Such a mechanism can choose its release point or its end-effector velocity but not both. Mechanisms containing 3 or more DOF with the correct kinematic structure are able to throw in R^3 and choose both the release point and the end-effector velocity simultaneously.

In recent work Mori et al. [4] has show his ability to control the translational velocity, angular velocity and direction in a 2-dimension plane independently with a single DOF mechanism. The only input is torque to the manipulator. The concept consists is to map the input torque that will change only one of the kinimatic variables and not the other two. This map is done over a given space and thus you can independently chose your translational and angular velocity as well as direction as long as it is in the valid search space. The manipulator and a search space example can be seen in Fig. 2.

Senoo et al.[5] used a torque controlled 3-DOF arm to create a high speed throwing trajectory. This arm falls into the *time and location insensitive* category of throwing.

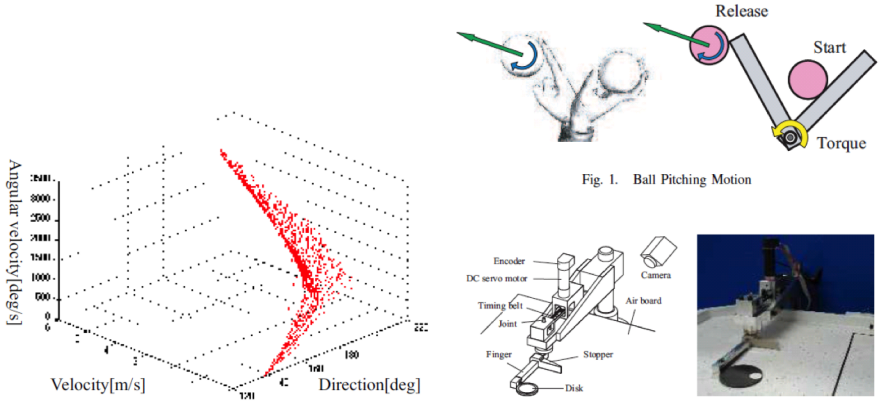
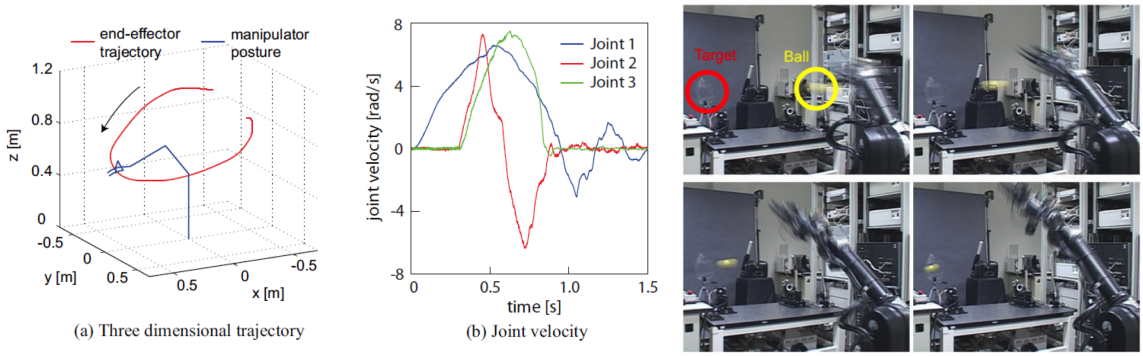


Fig. 1. Ball Pitching Motion

Figure 2: Map of the input torque that will change only one of the kinimatic variables and not the other two. This map is done over a given space and thus you can independently chose your translational and angular velocity as well as direction as long as it is in the valid search space.

Senoo used a kinematic chain approach based on how humans throw. Doing this Senoo was able to achieved an end-effector velocity of 6.0 m/s and can throw in R^3 space. This is done via the use of a planted robot arm made by Barret Technology Inc consisting of 3-DOF with a 360^o rotation base yaw actuator, see Fig. 3.



(a) Three dimensional trajectory

(b) Joint velocity

Figure 3: 3-DOF arm achieving an end-effector velocity of 6.0 m/s and can throw in R^3 space. This is done via the use of a planted robot arm made by Barret Technology Inc with a 360^o rotation base yaw actuator

Low degree of freedom throwing machines/robots are common. Typical throwing robots have between one and three degrees of freedom (DOF) [4, 6–9]. All of these mechanisms are limited to throwing in a plane.

These low degree of freedom throwing robots are either physically attached/planted to the mechanical ground or have a base that is significantly more massive then the arm.

Haddadin et al.[10] used their 7-DOF arm and a 6-DOF force torque sensor with standard feedback methods to dribble a basket ball. In addition Zhikun et al. [11] used reinforcement learning to teach their 7-DOF planted robot arm to play ping-pong. Likewise Schaal et al. [12] taught their high degree of freedom (30-DOF) humanoid to hit a tennis ball using an on-line special statistical learning methods. Visual feedback was used in the basketball throwing robot by Hu et al. [13] achieving accuracy of 99%. All of the latter robots were fixed to the ground to guarantee stability.

Kim et al. [14, 15] takes the research to the next level with finding optimal overhand and sidearm throwing motions for a high degree of freedom humanoid computer model. The model consists of 55-DOF and is not fixed to mechanical ground or a massive base. Motor torques are then calculated to create both sidearm and overhand throws that continuously satisfies the zero-moment-point stability criteria [16].

Balancing: Zero-Moment-Point (ZMP)

The past years of research in humanoids robotics has resulted in a stability criteria that must be followed for bipedal robots to stay stable. This is known as the Zero Moment Point criteria commonly referred to as ZMP [17]. ZMP is ubiquitous in the humanoid robotics community. The ZMP criteria states that a system is statically stable (balanced) if there is no moment acting on the connection between the end effectors touching the ground and the ground. This means that if the center of mass

is over the support polygon there will be no moment. The support polygon is defined by the are formed by connecting the out most portions of the end effectors (typically feet) that are touching the ground and/or walls, rails etc. If the zero moment point, the location of the center of mass (COM) projected in the direction of gravity, is located within this support polygon then the system is considered statically stable. Fig. 4 gives an example of the zero moment point on a bipedal robot in a single support phase and a double support phase.

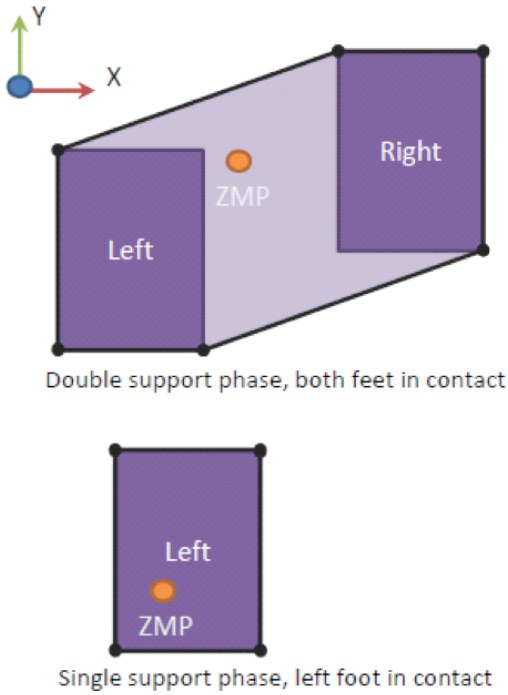


Figure 4: Example of the zero moment point on a bipedal robot in a single support phase (bottom) and a double support phase (top). If the zero moment point, the location of the center of mass (COM) projected in the direction of gravity, is located within this support polygon then the system is considered statically stable.

Single Support Phase: The single support phase of a bipedal robot is when a single foot is touching the ground. This creates a smaller support polygon.

Double Support Phase: The double support phase of a bipedal robot is when two feed of a bipedal robot are on the ground. This creates a larger support polygon. In addition there is a stable path that the ZMP can move from above one foot to the other. This allows the robot to guarantee stability while walking (static walking).

0.1.5 Vertical Leap

What constitutes a vertical leap and what needs to be done, i.e. where is it in this document.

- Higher archical system that keeps base functionality
- How to make the decisions
- Sensor integration into control
- Software structure that allows for multi-user implementation with a reduce risk of segfaulting
- Compliance: what is it and how do we fake it with our robot

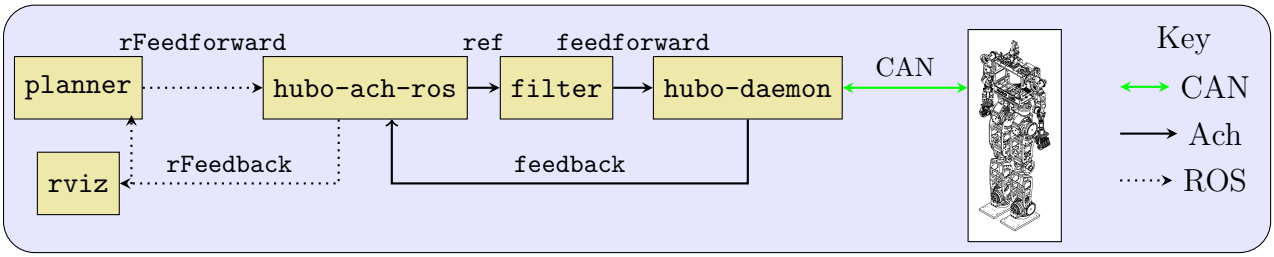


Figure 5: Feedback loop integrating Hubo-Ach with ROS

0.2 Hubo-Ach

The Hubo2+ is a 1.3m tall, 42kg full-size humanoid robot, produced by the Korean Advanced Institute of Science and Technology (KAIST) and spinoff company Rainbow Inc. [**?**]. It has 38 DOF: six per arm and leg, five per hand, three in the neck, and one in the waist. Sensors include three-axis force-torque sensors in the wrists and ankles, accelerometers in the feet, and an inertial measurement unit (IMU). The sensors and embedded motor controllers are connected via a Controller Area Network to a pair of Intel Atom PC104+ PCs.

Hubo-Ach [2] is an Ach-based interface to Hubo's sensors and motor controllers. This provides a conventional GNU/Linux programming environment, with the variety of tools available therein, for developing applications on the Hubo. It also efficiently links the embedded electronics and real-time control to popular frameworks for robotics software: ROS [**?**], OpenRAVE,[3] and MATLAB[4].

Reliability is a critical issue for software on the Hubo. As a bipedal robot, Hubo must constantly maintain dynamic balance; if the software fails, it will fall and break. A multi-process software design improves Hubo's reliability by isolating the critical balance code from other non-critical functions, such as control of the neck or arms.

[2] Available under permissive license, http://github.com/hubo/hubo-ach
[3] OpenRAVE: http://openrave.org/
[4] MATLAB: http://www.mathworks.com/

For the high-speed, low-latency communications and priority access to latest sensor feedback, Ach provides the underlying IPC.

Hubo-Ach handles CAN bus communication between the PC and embedded electronics. Because the motor controllers synchronize to the control period in a *phase lock loop* (PLL), the single `hubo-daemon` process runs at a fixed control rate and communicates on the bus. The embedded controllers lock to this rate and linearly interpolate between the commanded positions, providing smoother trajectories in the face of limited communication bandwidth. This communication process also avoids bus saturation; with CAN bandwidth of 1 Mbps and 200Hz control rate, `hubo-daemon` currently utilizes 78% of the bus. `Hubo-daemon` receives position targets from a `feedforward` channel and publishes sensor data to the `feedback` channel, providing the direct software interface to the embedded electronics.

Each Hubo-Ach controller is an independent processes. The controllers handle tasks such as balance, manipulation, and human-robot interaction. Each controller asynchronously reads state from the `feedback` Ach channel and sets reference positions in the `feedforward` channel. `Huobo-daemon` reads the most recent reference position from the `feedforward` channel on the the rising edge of its control cycle. This allows the controller processes to run at arbitrary rates without effecting the PLL of the embedded motor controllers or the CAN bus bandwidth utilization.

Figure 5 shows an example control loop integrating Hubo-Ach and ROS. The `hubo-daemon` communicates with the embedded controllers at 200Hz, publishing to the `feedback` channel. The `hubo-ach-ros` process bridges ROS topics and Ach channels. It translates messages on the `feedback` Ach channel to the `rFeedback` ROS topic and translates the `rFeedforward` ROS topic to the `ref` Ach channel. The `planner` process computes desired trajectories, which are relayed via `hubo-ach-ros` to `filter` for preprocessing to smooth the motion and reduce *jerk* before `hubo-daemon` com-

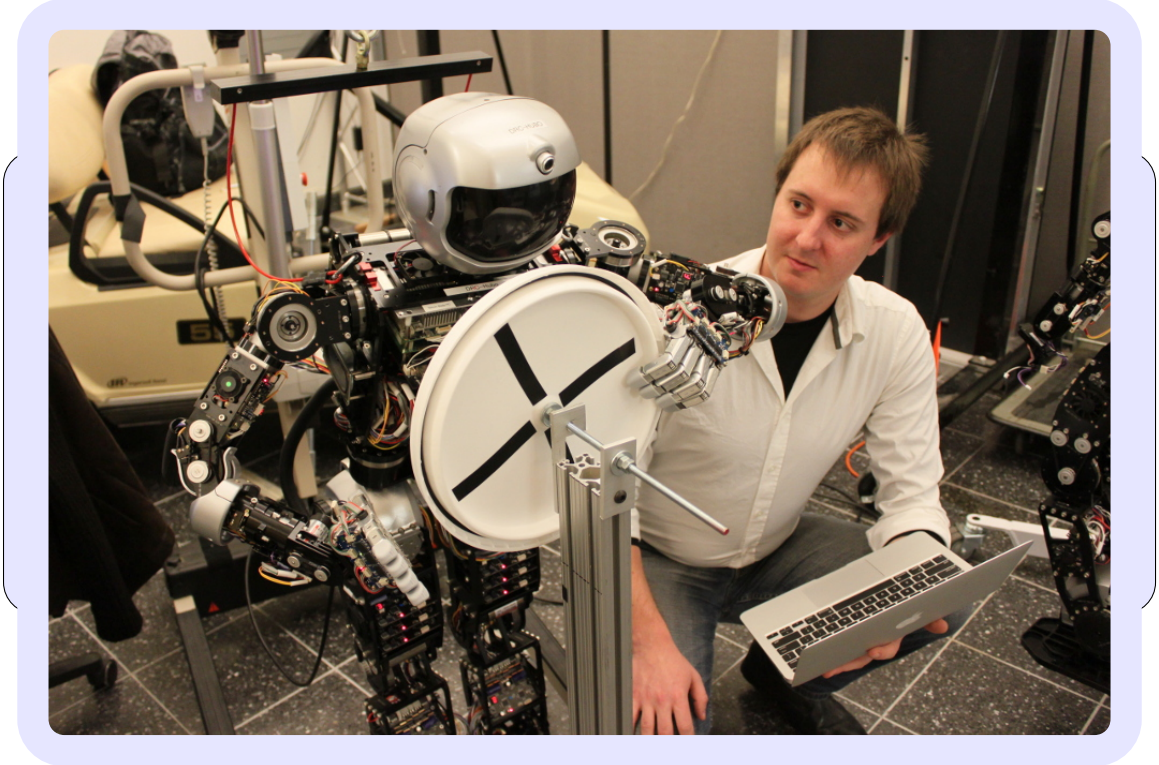


Figure 6: Hubo (left) turning a valve via Hubo-Ach alongside Daniel M. Lofaro (right). Valve turning developed in conjunction with Dmitry Berenson at WPI for the DARPA Robotics Challenge.

municates references to the embedded controllers. During operation, `rviz` displays a 3D model of the Hubo's current state. Each of these process runs asynchronously, communicating at different rates; however, `hubo-ach-daemon` maintains its 200Hz cycle, ensuring phase lock with the embedded controllers. This control loop effectively integrates real-time IPC and control under Hubo-Ach with the non-real-time ROS environment.

Hubo-Ach is in use for numerous projects at several research labs. Users include include groups at MIT, WPI, Ohio State, Purdue, Swarthmore College, Georgia Tech, and Drexel University. These projects primarily revolve around the DARPA Robot Challenge (DRC)[5] team DRC-Hubo[6]. The DRC includes rough terrain walking, ladder climbing, valve turning, vehicle ingress/egress and more. Figure 6 shows the Hubo using the Hubo-Ach system to turn a valve.

Hubo-Ach provides an effective base for developing real-time applications on the Hubo. Separating software modules into different processes increases system reliability. A failed process can be independently restarted, minimizing chance of damage to the robot. In addition, the controllers can run at fast rates because Ach provides high-speed low-latency communication with `hubo-daemon`. Hubo-Ach provides a C API that is easily called from high-level programming languages and integrates with popular platforms for robot software such as ROS and MATLAB, providing additional development flexibility. Hubo-Ach is a validated and easy to use interface between the mechatronics and the software control algorithms of the Hubo full-size humanoid robot.

[5]DARPA Robot Challenge: http://www.theroboticschallenge.org/
[6]DRC-Hubo Homepage: http://drc-hubo.com/

0.3 Contributions

0.3.1 Path Planning

0.4 RELATED WORK

Low degree of freedom throwing machines/robots are common. Typical throwing robots have between one and three degrees of freedom (DOF) [4, 6–9]. All of these mechanisms are limited to throwing in a plane. Sentoo et al.[5] achieved an end-effector velocity of 6.0 m/s and can throw in R^3 space using it's Barret Technology Inc 4-DOF arm with a 360^o rotation base yaw actuator. These low degree of freedom throwing robots are either physically attached/planted to the mechanical ground or have a base that is significantly more massive then the arm.

Kim et al. [15] takes the research to the next level with finding optimal over-arm and sidearm throwing motions for a high degree of freedom humanoid computer model. The model consists of 55-DOF and is not fixed to mechanical ground or a massive base. Motor torques are then calculated that both allows for a sidearm or overarm throw and continuously satisfies the zero-moment-point stability criteria[16].

0.5 METHODOLOGY

To create a valid throwing trajectory for a high-DOF, high-gain, position controlled robot, a desired line in R^3 in the direction of the desired velocity must be created. Each point in the line is temporally separated by the robot's command period T_r. All points in this line must be reachable. Each point in the line must have poses that do not create a self-collision. A valid throwing trajectory is created when the latter criteria are met.

0.5.1 Self-Collision Detection

Self-collision is an important when dealing with a high DOF robot. Unwanted self-collisions can cause permanent damage to the physical and electrical hardware as well as causing the robot not to complete the given task.

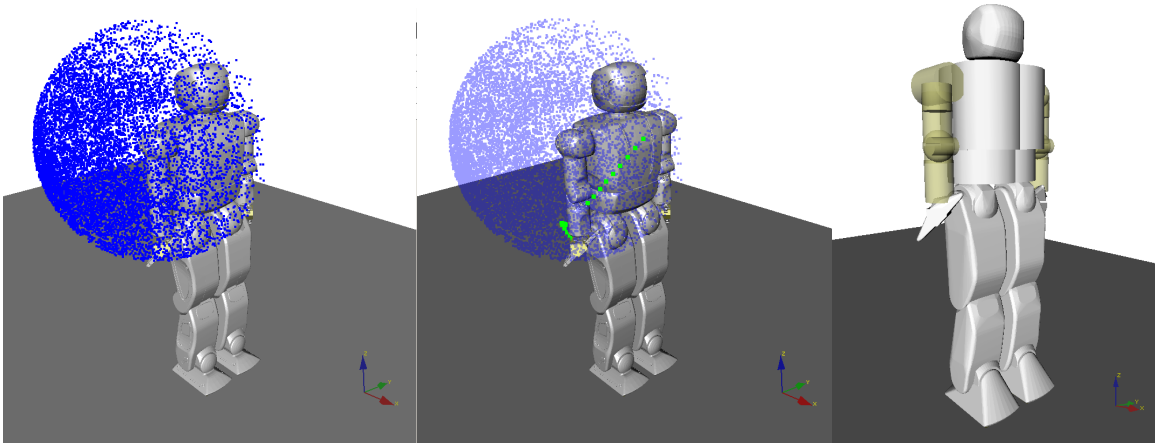


Figure 7: OpenRAVE model of Hubo KHR-4. Left: Model with SRM of right arm. Center: SRM (blue) with setup and velocity phase trajectories (green) Right: Collision Geometry

To aid in the detection of self-collisions a detailed model of the Hubo KHR-4 was made in the widely used open-source robot simulation environment OpenRAVE[18]. The model was created by exporting the three dimensional schematics that the physical robot was created with, to a format that OpenRAVE can use. This was done in order to ensure an accurate and detailed model. For these experiments we needed the external boundaries only; the internal geometry was replaced with a simplistic representation. The external shell is the only part now visible, see Fig 7 (Center). The Proximity Query Package (PQP) was used to detect collisions between any two pieces of the robot's external shell. Due to the high polygon count of the external shell

the computation time of detecting a collision was on the magnitude of seconds. It is advantageous to reduce this time if the system is to run live on the robot. Computation time is decreased significantly when boundary/collision geometries are simplified due to the lower polygon count. The collision geometries were further simplified to decrease computation time by making them primitives such as spheres, cylinder and boxes, see Fig 7 (Right).

Joint limitations are added to the model to mimic the physical robot. The model can be commanded the same configurations as the physical robot. A pose is commanded to the model, PQP searches for any collisions. With the simplified collision geometry self-collisions are detected on the order of milliseconds. If there are no collisions then the pose can be applied to the physical robot. A 5% increase in volume between the simplified collision geometry and the high polygon geometry was added to ensure all of the physical robot's movements will not collide due to minor calibration errors.

0.5.2 Reachable Area

The desired end-effector velocity must be achieved with all joint limits and self-collision constraints satisfied at all times. Typical methods of determining reachability is to move each joint through its full range of motion for each DOF[19, 20]. Due to the high DOF of the Hubo KHR-4 this method is not desirable. A sampling method described in this work is similar to Geraerts et al.[21]. It was used to accommodate the high DOF system. Both active and static joints must be defined to calculate the reachable area of a manipulator at a discrete time N. The static joints are assumed to hold a fixed position at time step N. Active joints are free to move to any position as long as it satisfies the joint angle limitations and does not create a self-collision. A uniform random number generator is used to assign each active joint with an angle

in joint space. Each random angle assigned is within the valid range of motion of the respective joint. The self-collision model described in Section 0.5.1 is used to determine the self-collision status with the randomly assigned joint angles. If there is no self-collision the end-effector position and transformation matrix T are calculated using forward kinematics.

$$\chi_{\mathbf{i}} = \begin{bmatrix} R_i & \Gamma_i \\ 0 & 1 \end{bmatrix} \tag{1}$$

$$T = [\chi_1 \cdot \chi_2 \cdot ... \cdot \chi_n] \tag{2}$$

where χ_i is the transformation between joint $i-1$ and i, R_i is the rotation of joint i with respect to joint $i-1$ and Γ_i is the translation of joint i with respect to joint $i-1$, and n is the number of joints in the kinematic chain.

The end-effector position and the joint angles used are recorded. This process is repeated multiple times to form a sparse representation of reachable end-effector positions in R^3 and the corresponding joint angles in joint space. The resulting representation is called the Sparse Reachable Map (SRM). Fig. 8 shows a cross section of the SRM about the right shoulder between -0.40 m to 0.40 m on X, -0.40 m to 0.40 m on Z, and -0.21 to -0.22 m on Y. The blue points show valid end-effector locations with known kinematic solution in joint space. Fig. 7 shows the SRM of the entire right arm. The SRM is used to calculate valid movement trajectories.

0.5.3 Trajectory Generation

An end-effector velocity, $\vec{V_e}$, is chosen based on target location, the well known equations of projectile motion, and the required velocity duration t_e. $\vec{V_e}$ must be held for a time span of t_e. The release point must be within the time span t_e. The

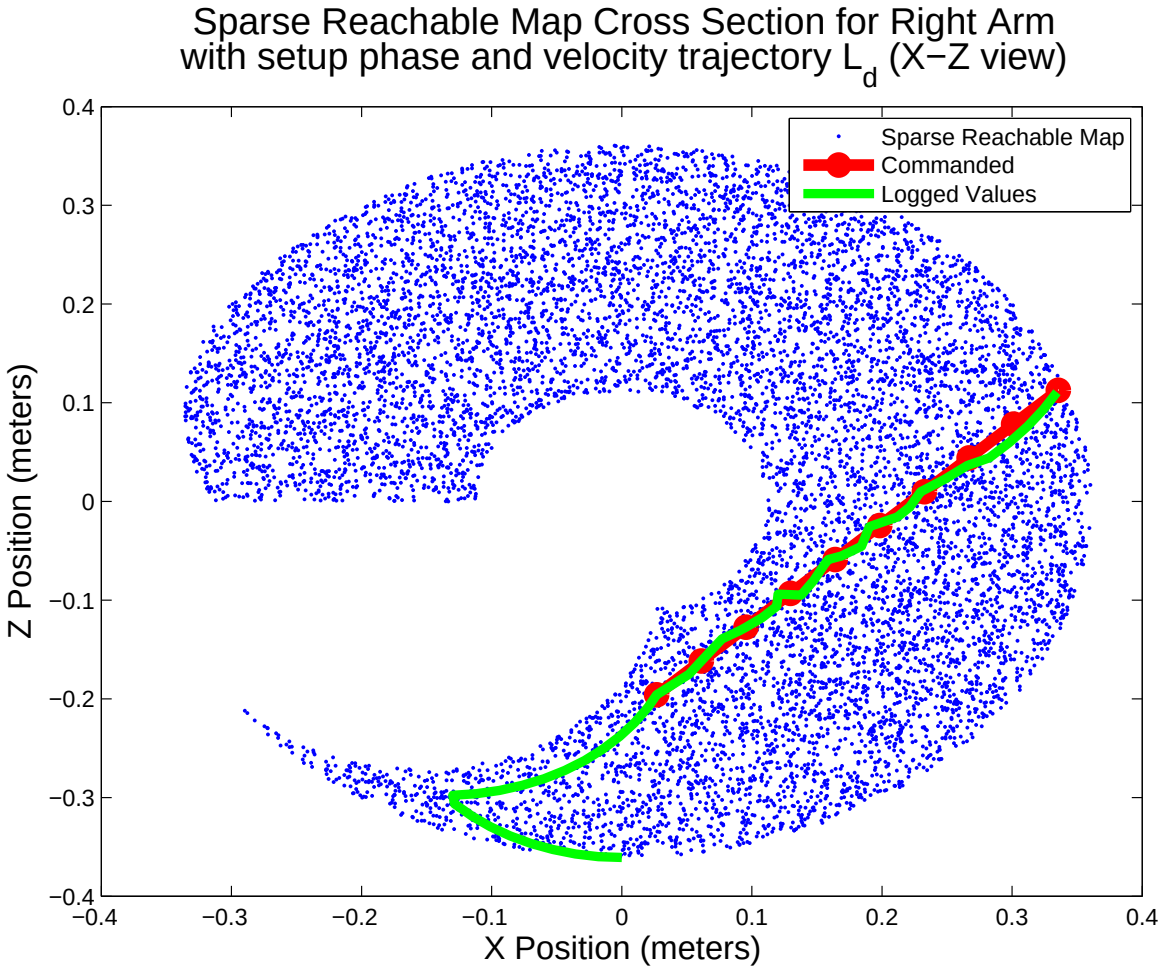


Figure 8: Cross section of the SRM about the right shoulder between -0.40 m to 0.40 m on X, -0.40 m to 0.40 m on Z, and -0.21 to -0.22 m on Y. (Blue) show valid end-effector locations with known kinematic solution in joint space. (Red) Commanded right arm end-effector position in R^3. (Green) The logged joint space values converted to R^3 using forward kinematics.

magnitude of the velocity in the direction of $\vec{V_e}$ immediately preceding time span t_e must be less than or equal to the magnitude of $\vec{V_e}$ during t_e. t_e must be an integer multiple of the robot's actuator command period T_r.

A line $\vec{l_d}$ in R^3 that passes through (X_0, Y_0, Z_0) in the direction of $\vec{V_e}$ is created. $\vec{L_d}$ is the discrete representation of $\vec{l_d}$. Each point in $\vec{L_d}$, (X_0, Y_0, Z_0), $(X_1, Y_1, Z_1) \cdots (X_n, Y_n, Z_n)$, are separated by a time span T_r.

The desired velocity is defined as

$$\vec{V_d} = [V_x\hat{i}, V_y\hat{j}, V_z\hat{k}] \tag{3}$$

The line $\vec{L_d}(n)$ is defined as

$$\vec{L_d}(n) = [X_n\hat{i}, Y_n\hat{j}, Z_n\hat{k}] \tag{4}$$

where n is the current zero based time step index value for the time span t_e. The change in $\vec{L_d}$ between time step 0 and n must be equal to our desired velocity $\vec{V_d}$.

$$\frac{\Delta\vec{L_d}|_0^n}{n \cdot T_r} = \vec{V_d} \tag{5}$$

thus

$$\vec{V_d} = \frac{\vec{L_d}(n) - \vec{L_d}(0)}{n \cdot T_r} \tag{6}$$

The line $\vec{L_d}$ at time step n can now be defined in terms of $\vec{V_d}$, T_r, the origin $\vec{L_d}(0)$, and the current zero based time step index value n.

$$\vec{L_d}(n) = n \cdot T_r \cdot \vec{V_d} + \vec{L_d}(0) \tag{7}$$

where

$$\vec{L}_d(0) = [X_0, Y_0, Z_0] \tag{8}$$

The line $\vec{L}_d$ is the trajectory the robot's end-effector must follow during the time span t_e. The starting point $\vec{L}_d(0)$ must be found so that $\vec{L}_d$ is within the reachable area. $\vec{L}_d(0)$ is set to a random starting points chosen within the SRM.

$$\vec{L}_d(0) \in SRM \tag{9}$$

All subsequent points in $\vec{L}_d$ must fall within some Euclidean distance d from any point in SRM. If one of the points in $\vec{L}_d$ fails this criteria a new random point is chosen for $\vec{L}_d(0)$ and the process is repeated.

Once an $\vec{L}_d$ is found that fits the above criteria the inverse kinematic solution must be found for each point and checked for reachability. Smaller values of d will increase the probability $\vec{L}_d$ is within the reachable area defined in the SRM however more iterations will be required to find a valid $\vec{L}_d$. Larger values of d will decrease the number of iterations needed to find a valid $\vec{L}_d$ however the probability of $\vec{L}_d$ being in the reachable area is decreased. In addition larger values of d decreases the system's ability to properly map near sharp edges in the SRM. Increasing the number of samples in the SRM will allow for larger values for d.

0.5.4 Inverse Kinematics

The trajectory $\vec{L}_d$ has one point with a known kinematic solution in R^3 and in joint space, $\vec{L}_d(0)$. The joint space kinematic solutions for points $\vec{L}_d(1) \rightarrow \vec{L}_d(n)$ are unknown. Mapping the robot's configuration $\vec{q} \in Q$ to the desired end-effector goal $\vec{x}_g \in X$, where Q is the robot's configuration space and X is in R^3, is done using Jacobian Transpose Controller used by Weghe et al.[22]. Weghe shows the Jacobian

as a linear map from the tangent space of Q to X and is expressed as

$$\dot{\vec{x}} = J\dot{\vec{q}} \tag{10}$$

The Jacobian Transpose method is used because of the high DOF of the Hubo KHR-4. Under the assumption of an obstacle-free environment the Jacobian Transpose Controller is guaranteed to reach the goal. A proof is shown by Wolovich et al.[23].

To drive the manipulator from its current position $\vec{x}$ to the goal positions $\vec{x}_g$ the error $\vec{e}$ is computed and the control law is formed.

$$\vec{e} = \vec{x}_g - \vec{x} \tag{11}$$

$$\dot{\vec{q}} = kJ^T\vec{e} \tag{12}$$

where k is a positive gain and self-collisions are ignored. The instantaneous motion of the end-effector is given by

$$\dot{\vec{x}} = J\dot{\vec{q}} = J(kJ^T\vec{e}) \tag{13}$$

The final pose $\vec{q}$ for our goal position $\vec{x}_g$ can now be found.

The Jacobian Transpose method works best when there is a small difference between the current position $\vec{x}$ and the goal position $\vec{x_g}$. $\vec{L}_d(0)$ is known both in X and in Q and is the starting point.

$$\vec{x} = \vec{L}_d(0) \tag{14}$$

$$\vec{q}_0 = SRM\left(\vec{L}_d(0)\right) \tag{15}$$

The goal position $\vec{x}_g$ is set to the next point in $\vec{L}_d$

$$\vec{x}_g = \vec{L}_d(1) \tag{16}$$

The pose $\vec{q}_1$ can now be calculated

$$\vec{q}_1 = \vec{q}_0 + \dot{\vec{q}}_0 = \vec{q}_0 + kJ^T\vec{e}|_{\vec{x}}^{\vec{x}_g} \tag{17}$$

where $\vec{x}_g = \vec{L}_d(0)$ and $\vec{x} = \vec{L}_d(1)$. $\vec{L}_d(1)$ is now known both in X and in Q. Now $\vec{x} = \vec{L}_d(1)$ and the process is repeated until all points in $\vec{L}_d$ are known both in X and Q.

0.5.5 On-Line Trapezoidal Motion Profile

The robot's starting position $\vec{x}_0$ is not guaranteed to be the same as the first point in the velocity trajectory $\vec{L}_d$. To avoid over large accelerations when giving this step input from $\vec{x}_0$ to $\vec{L}_d(0)$ an on-line trapezoidal motion profile (TMP) was used to generate joint space commands with the desired limited angular acceleration and velocity. The TMP was only active during the setup phase where the robot's end-effector moves from $\vec{x}_0$ to $L_d(0)$. This is because the TMP's inherent nature has the potential to adversely effect the desired velocity in R^3 under high angular velocity and acceleration conditions in joint space.

The TMP was designed to limit the applied angular velocity and acceleration in joint space and to prevent over-current/torque. An important advantage over simply limiting output velocity and acceleration is that the TMP has little to no overshoot. When a clipped and rate-limited velocity profile is integrated, the resulting position

trajectory may over or undershoot due to this non-linear system behavior. The TMP accounts for the imposed limits inherently, and will arrive at a static goal without overshoot. Table 1 describes the three regions that make up the TMP.

Table 1: Trapezoidal Motion Profile Regions

Region 1	Accelerate at maximum acceleration in direction of goal
Region 2	Achieve and hold maximum velocity
Region 3	Decelerate to zero velocity to reach goal

The area under the velocity trapezoid in region 1-3 is the total displacement achieved by the profile. By shaping this profile based on initial and goal conditions, any goal position can be precisely reached, even if velocity clipping occurs. The shape of the profile can be challenging to identify, since it is not always a trapezoid. For large velocity and acceleration limits and small displacements, the profile will only reach a fraction of maximum velocity, and will be triangular. The varying shape of the profile means that calculating and storing complete motion profiles for each update may be required. This paper's method removes the need for complete profile generation and storage.

Regions one and two of the velocity profile are bounded by the maximum acceleration, a_m, and maximum velocity, v_m, respectively. In these regions the joint moves towards the goal as fast as the limits allow. In region three the joint has reached a deceleration distance d_s from the goal. It now accelerates at $-a_m$. When the velocity reaches zero, the joint has exactly arrived at the goal position. d_d is the integral of the velocity profile in region three, given by (18).

As long as the distance to the goal d_g and d_s are equal then the controller needs to decelerate at the maximum rate to come to rest at the goal. Conversely, for the

current goal distance, there is a critical velocity v_c such that, if the joint began moving at this velocity in the following time-step τ, it could decelerate at a_m to reach the position goal. The controller minimizes the error between v_c and v_0 at each time-step.

Since the joint is moving with velocity v_0 during a current time-step, some initial distance d_i (19) is traveled before the joint can be affected. Defining $\hat{u}$ as the sign of the distance to the goal, v_c is related to d_g and d_i quadratically in (21). This equation assumes simple trapezoidal integration. Solving for v_c using the quadratic formula generally produces complex roots due to the possibility of negative v_0 or d_g. In (22), $v_0 \cdot \hat{u}$ is the current velocity relative to the goal direction, producing a positive term if the signs of both terms match. This result will always produce a real value for v_0 and d_g.

$$d_s = \frac{v_0^2 sign(v_0)}{2a_m} \tag{18}$$

$$d_i = v_0\tau + \frac{v_c - v_0}{2}\tau \tag{19}$$

$$\hat{u} = sign(d_g) \tag{20}$$

$$v_c^2 = 2a_m\left(d_g - d_s\right) \tag{21}$$

$$v_c = \hat{u}a_m\left(\sqrt{\frac{a_m\tau^2 - 4\hat{u}v_0\tau + 8|d_g|}{4a_m}} - \frac{\tau}{2}\right) \tag{22}$$

0.5.6 Throwing

0.6 Introduction

In early February 2012 the director of the Philadelphia Science Festival asked the Drexel Autonomous Systems Lab (DASL)[7] if they could have their full-size humanoid Jaemi Hubo throw the ceremonial first pitch at the second annual *Science Night at the Ballpark*. On April 28th, 2012 Hubo successfully threw the first pitch at the Philadelphia Phillies vs. Chicago Cubs game, see Fig. 9. According to the USA Today were 45,196 fans at the game and thousands more were watching it on television.

Hubo was the first full-size humanoid to throw the inaugural pitch at a Major League Baseball game. This task poses challenges in the area of fully-body locomotion, coordination and stabilization that must be addressed. This paper describes how the latter was done via the analyses/tests of three different approaches and the resulting final design. Section **??** gives a brief introduction to work already done in the field as well as states the requirements for the pitch. Section 0.7 describes the three different methods tested where: Section 0.7.1 discusses the balancing methods and criteria used. Section 0.7.2 describes the human-robot kinematic mapping approach that uses a motion capture system to capture a human's throwing motion then mapping that to a full-size humanoid. Section 0.7.3 describes a fully automated approach that uses the sparse reachable map (SRM) to provide viable full body throwing trajectories with the desired end effector velocity[24]. Section 0.7.4 describes the final method explored which is based on key-frame trajectories. Section **??** compares the tests and analyses of each of the methods. Section **??** describes the finial design in detail and the modifications needed to make the robot's pitch reliable. Finally Section **??** gives final thoughts and possible improvements for future years.

[7]Drexel Autonomous Systems Lab: http://dasl.mem.drexel.edu

Figure 9: Hubo successfully throwing the first pitch at the second annual Philadelphia Science Festival event Science Night at the Ball Park on April 28th, 2012. The game was between the Philadelphia Phillies and the Chicago Cubs and played at the Major League Baseball stadium Citizens Bank Park. The Phillies won 5-2. Video of the pitch can be found at http://danlofaro.com/Humanoids2012/#pitch

0.7 Methodology

0.7.1 Balance and Stability

Each of the methods used have to be stable through the motion in order for the system to be stable (i.e. not to fall down). The well known zero-moment-point (ZMP) criteria is what each method must adhere to in order to stay statically stable[25]. To handle perturbation an active balance controller was added. The active balance

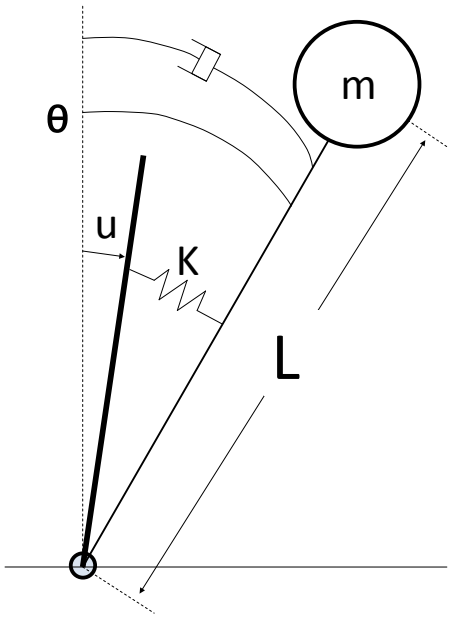


Figure 10: Hubo modeled as a single inverted pendulum with COM located a distance L from

controller is applied on top of the pre-defined trajectories. Hubo is modeled as a single inverted pendulum with the center of mass (COM) located at length L from the ankle. The compliance of the robot is composed of a spring K and a damper C, see Fig. 10. An IMU located at the COM gives the measured orientation.

The dynamic equation of the simplified model is assumed to be the same in both the sagittal and coronal plane.

$$mL^2\ddot{\theta} + C\dot{\theta} - K\theta = Ku \tag{23}$$

This can be linearized and made into the transfer function:

$$G(s) = \frac{\Theta(s)}{U(s)} = \frac{\frac{K}{mL^2}}{s^2 + \frac{C}{mL^2}s + \frac{K-mgL}{mL^2}} \tag{24}$$

Prior work on the model and controller for the Hubo by Cho et. al. calculated

K=753 $\frac{Nm}{rad}$ and C=18 $\frac{Nm}{sec}$ using the free vibration response method[26].

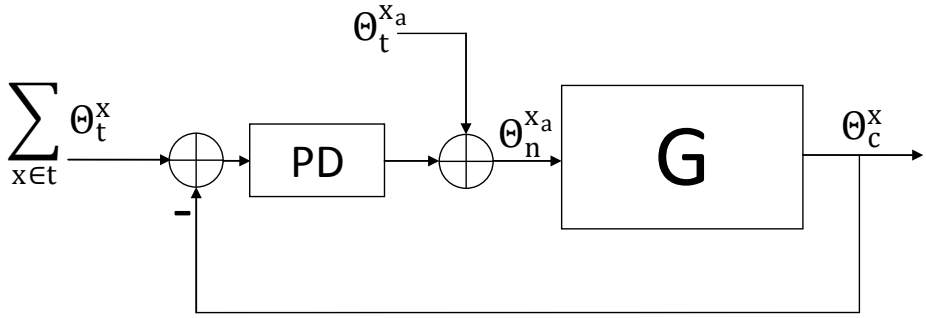


Figure 11: Block diagram of the balance controller used to balance Hubo in this work.

The control law is as follows

$$\theta_n^{x_a} = \theta_t^{x_a} + \left(K_p^x + sK_d^x\right)\left(\sum_{x \in t} \theta_t^x - \theta_c^x\right) \tag{25}$$

Where θ_t is the desired trajectory of the lower body (pitch or roll), x denotes pitch or roll and x_a denotes pitch or roll on the ankle. θ_c is the orientation of the center of mass in the global frame. θ_n is the resulting trajectory. K_p and K_d are the proportional and derivative gains. The resulting control allows for a stable stance even with perturbations from upper body motions.

0.7.2 Human to Humanoid Kinematic Mapping

Motion capture (MoCap) systems are commonly used to record high degree of freedom human motion. Athletic trainers in baseball, football and cycling use motion capture to analyze and improve throwing and lower limb motions[27–30]. MoCap systems are also used to generate human-like motions and map those motion to humanoids[31, 32]. Fig. 12 shows the Hubo's kinematic structure (left) and the

human (MoCap) kinematic structure(left). The human has 3-DOF at each joint while the humanoid has limited DOF at each corresponding joint. Some of the challenges in mapping between the human kinematic structure (from MoCap) to a humanoid's kinematic structure are:

- The difference in the total degree of freedom (DOF).
- The difference in the kinematics descriptions.
- The different Kinematic constraints.

Gaertner et. al.[33] uses an intermediate model (Master Motor Map) to decouple motion capture data for further post-processing tasks. Our approach is to: a) Chose a set MoCap model. b) Preform motions where the pitch motions are decoupled (roll and yaw stays constant), avoids singularities and robot joint position limitations. c) Combine joint values for near by joints (reduce the model to the same DOF as the robot). d) Some tests require the addition of static offsets to joints to ensure the zero-moment-point (ZMP) criteria is satisfied as stated in Section 0.7.1

To test this method we used a human subject to throw a ball using upper and lower body movements. All motions were in the sagittal plane to keep pitch joints decoupled. To avoid the robot's joint limit of $\pm 180^o$ an underhand throwing motion was used. Fig. 13 shows the human throwing the ball and the robot throwing the ball to the mapped motion of the human.

To ensure balance throughout the motion the balance controller as described in Section 0.7.1 was applied and the static ZMP criteria was checked for the entire trajectory. The human subject threw the ball approximately eight feet (244 cm). The mapping of the latter motion caused the robot to throw the ball approximately five feet (152 cm). The discrepancy comes from the proportional difference in limb

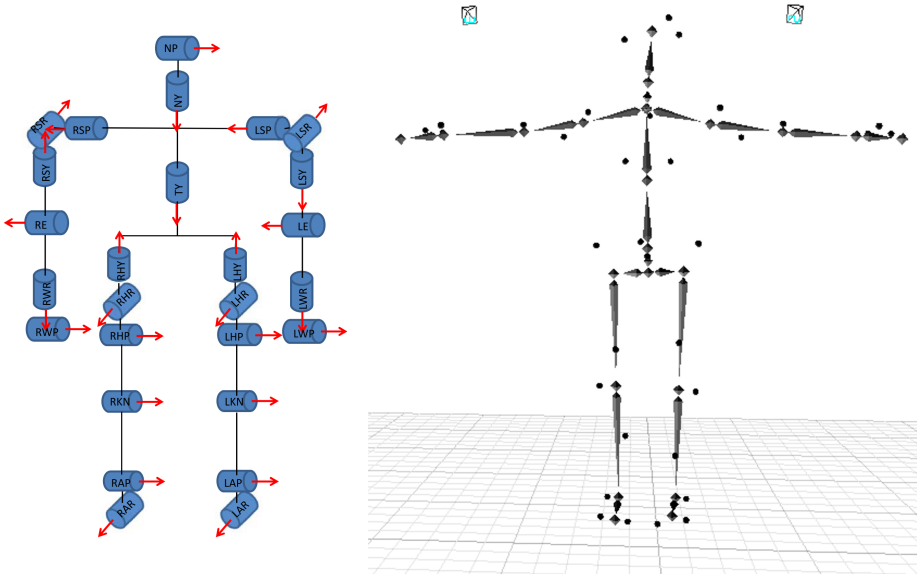


Figure 12: Left: Jaemi Hubo joint order and orientation using right hand rule. Right: Motion capture model of human figure

length from the human to the robot. A side by side video of the human and the robot throwing the ball is available for viewing on the this papers's homepage[8].

0.7.3 Throwing Using Sparse Reachable Map

A Sparse Reachable Map (SRM) is used to create a collision free trajectories while having the end-effector reach a desired velocity as described in Lofaro et. al.[24]. The SRM has been shown to be a viable method for trajectory generation for high degree of freedom, high-gain position controlled robots. This remains true when operating without full knowledge of the reachable area as long as a good collision model of the robot is available. The end-effector velocity (magnitude and direction) is specified as well as a duration of this velocity. The SRM is created by making a sparse map of the reachable end-effector positions in free space and the corresponding poses in joint space by using random sampling in joint space and forward kinematics. The desired

[8]MoCap to Robot (Video): http://danlofaro.com/Humanoids2012/#mocap

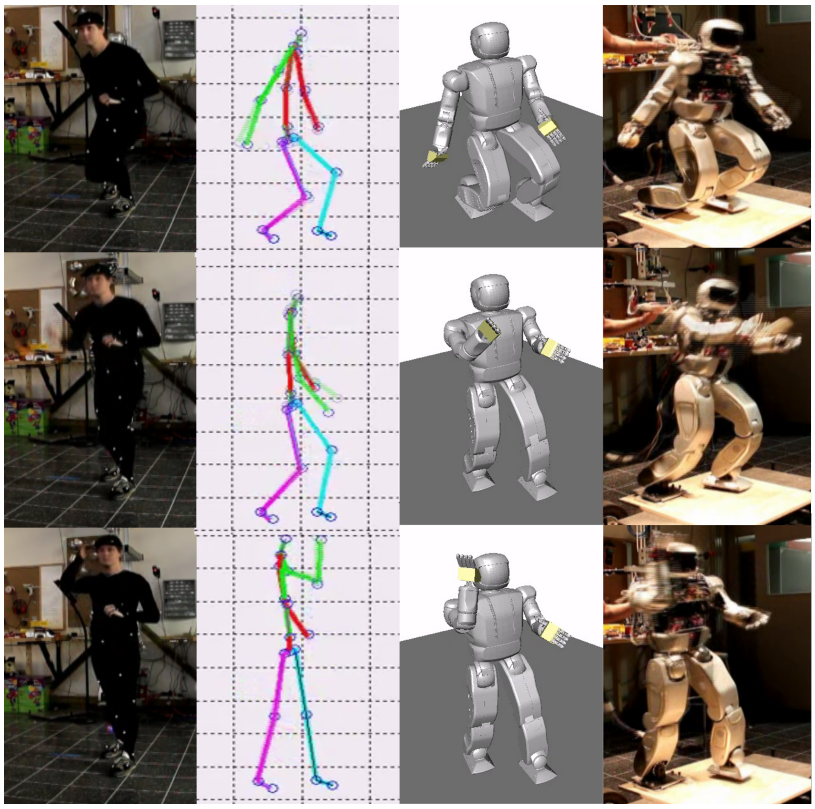


Figure 13: (Left to Right): (1) Human throwing underhand in sagittal plane while being recorded via a motion capture system. (2) Recorded trajectory mapped to high degree of freedom model. (3) High degree of freedom model mapped to lower degree of freedom OpenHUBO. (4) Resulting trajectory and balancing algorithm run on Hubo.[34]

trajectory in free space is placed within the sparse map with the first point of the trajectory being a known pose from the original sparse map.

$$L_d(0) \in SRM \tag{26}$$

$L_d(0)$ is known both in joint space and in free space. The Jacobian Transpose Controller method of inverse kinematics as described by Wolovich et al.[23] is then used to find the subsequent joint space values for the free space points in the trajectory.

$$q_1 = q_0 + \dot{q}_0 = q_0 + kJ^T e|_{x_0}^{x_1} \tag{27}$$

Where q_0 and x_0 is the current pose and corresponding end-effector position respectively. q_1 is the next pose for the next desired end-effector position x_1. Each desired end-effector position x must be within a euclidean distance d (user defined) from any point in the SRM.

$$min\left(|x - SRM|\right) < d \tag{28}$$

If one of the points in x fails this criteria a new random point is chosen for $L_d(0)$ and the process is repeated.

Each pose in the trajectory is checked against the collision model to guarantee no self-collisions. The collision model is based on the OpenRAVE model of the Hubo platform called OpenHUBO, see Fig 14.

The commanded trajectory produces the desired velocity of 4.9 m/s at 60^o. This was then tested on the OpenHUBO and on the Jaemi Hubo platform, Fig 15 and Fig 16 respectively.

To ensure balance throughout the motion the balance controller as described in Section 0.7.1 was applied and the static ZMP criteria was checked for the entire trajectory. This method worked as desired. In approximately 10% of the tests one or more joints would over torque and shutdown. This is due to the system not taking the robots power limitations into account.

0.7.4 Key-Frame Motion

Key-frame motion profiles for humanoids borrows from the animation industries' long used techniques. When making an animation the master artist/cartoonist will

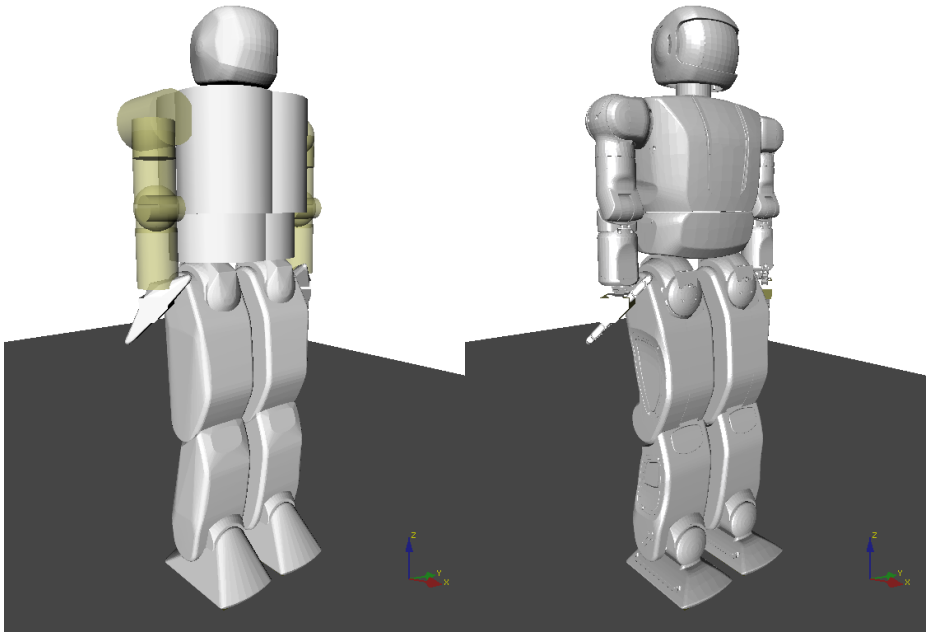


Figure 14: OpenHUBO - OpenRAVE model of Hubo KHR-4. Left: Collision Geometry. Right: Model with protective shells[24].

create the character in the most important (or key) poses. The apprentice will draw all of the frames between the key poses. We borrowed this technique when we: posed the robot in the desired pose, record the values in joint space, and make a smooth motion between poses. In place of the apprentice, forth order interpolation methods were used to make smooth trajectories between poses. Forth order interpolation was used in order to limit the jerk on each of the joints. The resulting trajectory is a smooth well defined motion as seen in Fig. 17.

To ensure stability throughout the motion the balance controller as described in Section 0.7.1 was applied and the static ZMP criteria was checked for the entire trajectory. The resulting end effector velocity was 4.8 $\frac{m}{s}$ at the release point. Fig. 18 shows the plot of the magnitude of the end effector's velocity. It should be noted that at the instance of release the velocity vector is at an elevation of 40^o from the ground.

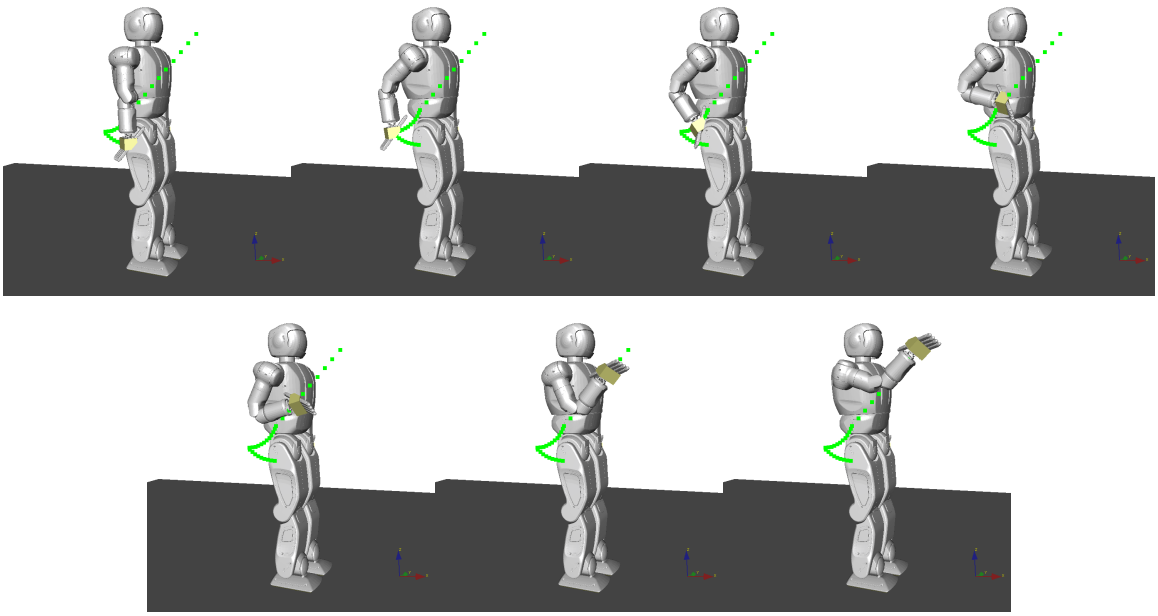


Figure 15: OpenHUBO running the throwing trajectory immediately after the setup phase is completed. x_0 is top left. Frames are read left to right and have a Δt of 0.15s[24]

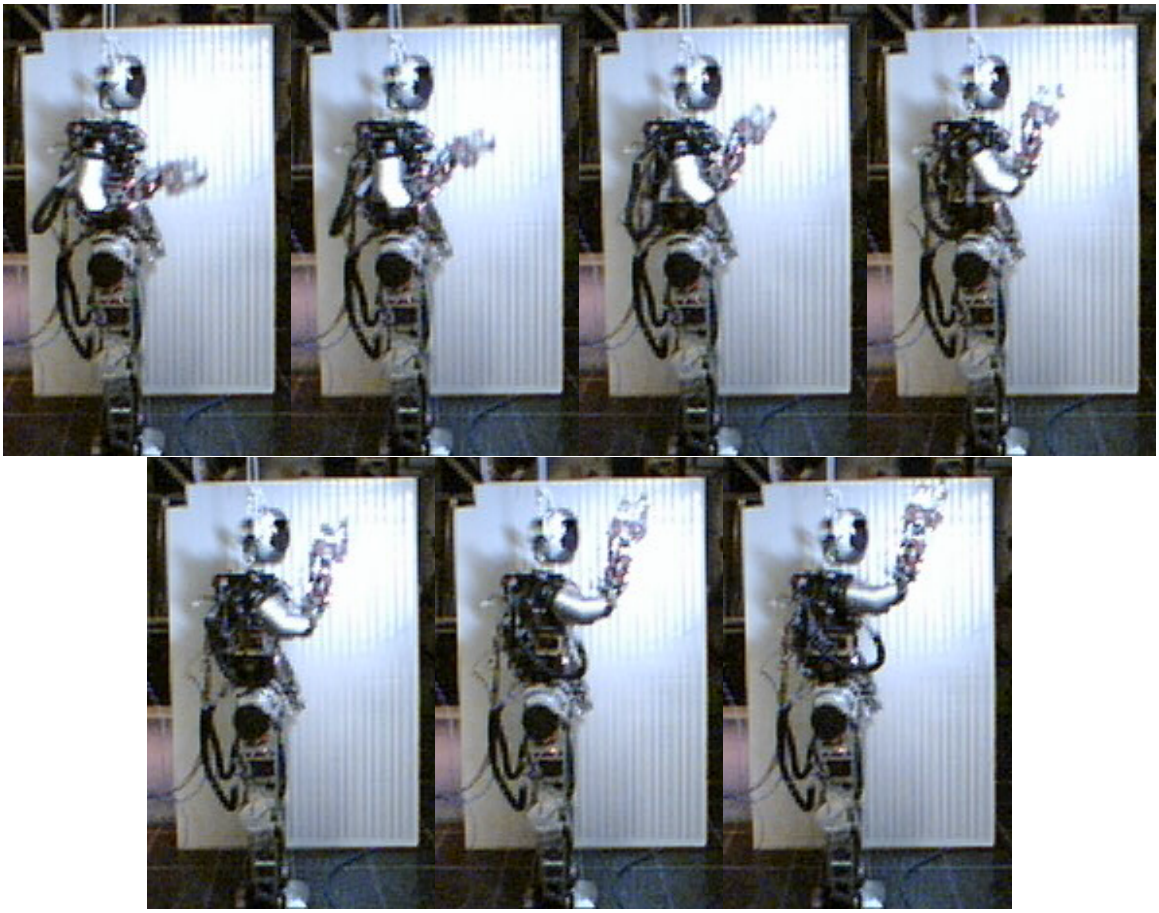


Figure 16: Jaemi Hubo running the throwing trajectory immediately after the setup phase is completed. x_0 is top left. Frames are read left to right and have a Δt of 0.15 sec[24]

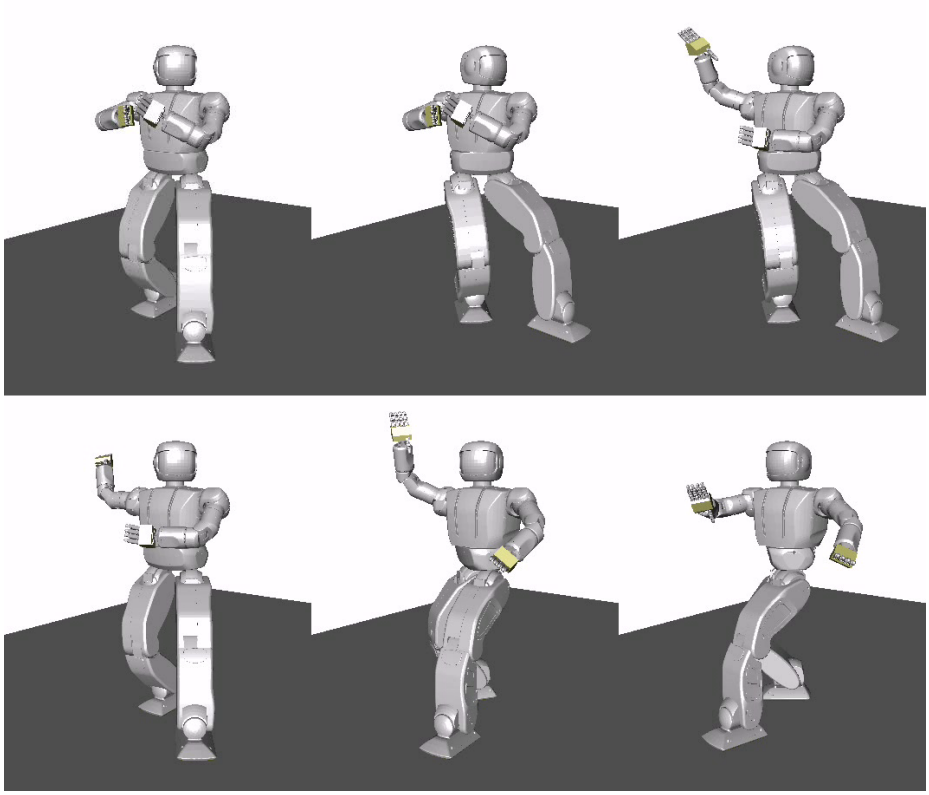


Figure 17: OpenHUBO using key-frame based method for throwing trajectory creation. Frames are read from top left to bottom right. Video of the above trajectory can be found at http://danlofaro.com/Humanoids2012/#keyframe

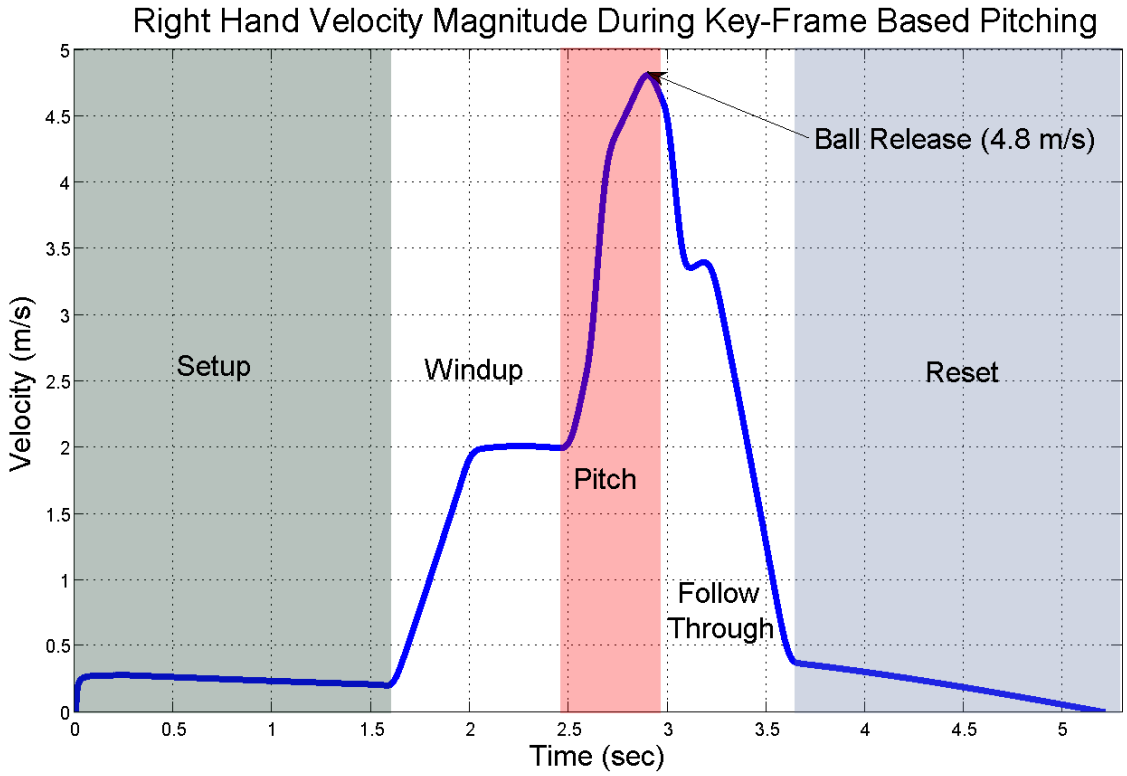


Figure 18: Velocity vs. Time graph showing the magnitude of the end-effector's velocity for the key-frame based throwing motion. The six different stages of pitching are also shown. Setup: move from the current position to th throw stance. Windup: end effector starts to accelerate from the throw stance and move into position for the start of the pitch state. Pitch: end effector accelerates to release velocity. Ball Release: the ball leaves the hand at maximum velocity (4.8 $\frac{m}{s}$) at an elevation of 40^o from the ground. Follow Through: reducing velocity of end effector and all joints. Reset: moves to a ready state for anther throw if needed.

0.8 Experiment

0.8.1 Task

Turning a valve with whole body (more of a lever)

- Find the valve - sensing
- Move to the valve - close loop on sensing
- Grab the valve
- Jump on the valve

0.8.2 System setup

State diagram of plan

- Find the valve - sensing
- Move to the valve - close loop on sensing
- Grab the valve
- Jump on the valve

Software structrue

- Hubo-Ach
- ROS
- Etc.

Sensors Chosen

- FT

- IMU
- Monocular
- RGB-D

Controllers

- Walking
- Balance
- Complience
- IK
- Visual Servoing

0.8.3 Results

- What worked
- What did not
- To be improved

0.9 Conclusions

- How did the system work well
- Insights as to what needs to change for the next vertical leap
- Future work

Bibliography

[1] R. Ellenberg, R. Sherbert, P. Oh, A. Alspach, R. Gross, and J. Oh, "A common interface for humanoid simulation and hardware," in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS Int'l Conference on*, pp. 587 –592, Dec 2010.

[2] R. Ellenberg, D. Grunberg, P. Oh, and Y. Kim, "Using miniature humanoids as surrogate research platforms," in *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS Int'l Conference on*, pp. 175 –180, Dec 2009.

[3] Y. Jun, R. Ellenberg, and P. Oh, "Realization of miniature humanoid for obstacle avoidance with real-time zmp preview control used for full-sized humanoid," in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS Int'l Conference on*, pp. 46 –51, Dec 2010.

[4] W. Mori, J. Ueda, and T. Ogasawara, "1-dof dynamic pitching robot that independently controls velocity, angular velocity, and direction of a ball: Contact models and motion planning," in *Robotics and Automation, 2009. ICRA '09. IEEE Int'l Conference on*, pp. 1655 –1661, May 2009.

[5] T. Senoo, A. Namiki, and M. Ishikawa, "High-speed throwing motion based on kinetic chain approach," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ Int'l Conference on*, pp. 3206 –3211, Sept 2008.

[6] N. Kato, K. Matsuda, and T. Nakamura, "Adaptive control for a throwing motion of a 2 dof robot," in *Advanced Motion Control, 1996. AMC '96-MIE. Proceedings., 1996 4th Int'l Workshop on*, vol. 1, pp. 203 –207 vol.1, Mar 1996.

[7] K. M. Lynch and M. T. Mason, "Dynamic nonprehensile manipulation: Controllability, planning, and experiments," *Int'l Journal of Robotics Research*, vol. 18, pp. 64–92, 1997.

[8] T. Nakamura, "Search guided by skill in motion planning using dynamic programming," in *Advanced Motion Control, 1996. AMC '96-MIE. Proceedings., 1996 4th Int'l Workshop on*, vol. 2, pp. 711 –716 vol.2, Mar 1996.

[9] A. Sato, O. Sato, N. Takahashi, and M. Kono, "Trajectory for saving energy of a direct-drive manipulator in throwing motion," *Artificial Life and Robotics*, vol. 11, pp. 61–66, 2007.

[10] S. Haddadin, K. Krieger, M. Kunze, and A. Albu-Schaffer, "Exploiting potential energy storage for cyclic manipulation: An analysis for elastic dribbling with an anthropomorphic robot," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ Int'l Conference on*, pp. 1789 –1796, Sept 2011.

[11] Z. Wang, C. H. Lampert, K. Mulling, B. Scholkopf, and J. Peters, "Learning anticipation policies for robot table tennis," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ Int'l Conference on*, pp. 332 –337, Dec 2011.

[12] S. Schaal, S. Vijayakumar, S. D'Souza, A. Ijspeert, and J. Nakanishi, "Real-time statistical learning for robotics and human augmentation," in *Int'l Symposium of Robotics Research (ISRR01)*, pp. 117–124, Springer, 2001.

[13] J.-S. Hu, M.-C. Chien, Y.-J. Chang, S.-H. Su, and C.-Y. Kai, "A ball-throwing robot with visual feedback," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ Int'l Conference on*, pp. 2511 –2512, Oct 2010.

[14] J. Kim, "Motion planning of optimal throw for whole-body humanoid," in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS Int'l Conference on*, pp. 21 –26, Dec 2010.

[15] J. H. and Kim, "Optimization of throwing motion planning for whole-body humanoid mechanism: Sidearm and maximum distance," *Mechanism and Machine Theory*, vol. 46, no. 4, pp. 438 – 453, 2011.

[16] M. Vukobratovic, "How to control artificial anthropomorphic systems," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 3, pp. 497 –507, sept. 1973.

[17] M. Vukobratovic and B. Borovac, "Zero-moment point - thirty five years of its life," *I. J. Humanoid Robotics*, vol. 1, no. 1, pp. 157–173, 2004.

[18] R. Diankov, *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, Aug 2010.

[19] Z.-Y. Ying, Y.-G. Xi, and Z.-H. Zhang, "Test of the reachability of a robot to an object," in *Robotics and Automation, 1989. Proceedings., 1989 IEEE Int'l Conference on*, pp. 490 –494 vol.1, May 1989.

[20] Z. Xue and R. Dillmann, "Efficient grasp planning with reachability analysis," in *Intelligent Robotics and Applications* (H. Liu, H. Ding, Z. Xiong, and X. Zhu, eds.), vol. 6424 of *Lecture Notes in Computer Science*, pp. 26–37, Springer Berlin / Heidelberg, 2010.

[21] R. Geraerts and M. Overmars, "Reachability analysis of sampling based planners," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE Int'l Conference on*, pp. 404 – 410, Apr 2005.

[22] M. Vande Weghe, D. Ferguson, and S. Srinivasa, "Randomized path planning for redundant manipulators without inverse kinematics," in *Humanoid Robots, 2007 7th IEEE-RAS Int'l Conference on*, pp. 477 –482, 29 2007-dec. 1 2007.

[23] W. A. Wolovich and H. Elliott, "A computational technique for inverse kinematics," in *Decision and Control, 1984. The 23rd IEEE Conference on*, vol. 23, pp. 1359 –1363, dec. 1984.

[24] D. Lofaro, R. Ellenberg, P. Oh, and J. Oh, "Humanoid throwing: Design of collision-free trajectories with sparse reachable maps," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ Int'l Conference on*, Oct 2012.

[25] M. Vukobratovic and J. Stepanenko, "On the stability of anthropomorphic systems," *Mathematical Biosciences*, vol. 15, no. 12, pp. 1 – 37, 1972.

[26] B.-K. Cho, S.-S. Park, and J. ho Oh, "Controllers for running in the humanoid robot, hubo," in *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS Int'l Conference on*, pp. 385 –390, Dec 2009.

[27] S. Fleisig, R. F. Escamilla, J. R. Andrews, T. Matsuo, Y. Satterwhite, and S. W. Barrentine, "Kinematic and kinetic comparison between baseball pitching and football passing," *Journal of Applied Biomechanics*, vol. 12, pp. 207–224, 1996.

[28] W. Barrentine, T. Matsuo, R. F. Escamilla, G. S. Fleisig, and J. R. Andrews, "Kinematic analysis of the wrist and forearm during baseball pitching," *Journal of Applied Biomechanics*, vol. 14, pp. 24–39, Jan 1998.

[29] Y. Mochizuki, T. Matsumoto, S. Inokuchi, and K. Omura, "Computer simulation of the effect of ball mass and shape to upper limb in baseball pitching," *Theoretical and Applied Mechanics*, vol. 47, pp. 283–292, 1998.

[30] A. Uesaki, Y. Mochizuki, T. Matsuo, K. Hashizume, K. Omura, and S. Inokuchi, "Computer simulation for dynamics analysis of pedaling motion on lower limbs in a racing cycle," *Theoretical and Applied Mechanics*, vol. 48, pp. 197–205, 1999.

[31] Q. Huang, Z. Peng, W. Zhang, L. Zhang, and K. Li, "Design of humanoid complicated dynamic motion based on human motion capture," in *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ Int'l Conference on*, pp. 3536 – 3541, Aug 2005.

[32] S.Pollard, J. Hondgins, M.J.Riley, and C. Atkeson, "Adapting human motion for the control of a humanoid robot," in *In Proc. of IEEE Int'l Conference on Robotics and Automation*, 2002.

[33] S. Gaertner, M. Do, T. Asfour, R. Dillmann, C. Simonidis, and W. Seemann, "Generation of human-like motion for humanoid robots based on marker-based motion capture data," *Robotics (ISR), 2010 41st Int'l Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pp. 1 –8, Jun 2010.

[34] D. Lofaro and P. Oh in *Humanoid Throws Inaugural Pitch at Major League Baseball Game: Challenges, Approach, Implementation and Lessons Learned*, nov. 2012.