Motion Capture Analysis of Diabolo Juggling Report

Jeb Cui

Thomas Jefferson High School for Science and Technology

Computer Systems Research Lab

Motion Capture Analysis of Diabolo Juggling Report

Researchers have explored modeling dynamical systems involving juggling before, like ball, flower stick, or yoyo juggling. However, an area of modeling juggling that has recently emerged is diabolo juggling. As shown in Figure 1, a diabolo, or a Chinese yoyo, is similar to what a hand-size yoyo would be. However, there are several differences: the string and diabolo are not attached, there is a pair of sticks to aid in controlling the diabolo, and the diabolo is distinctly hour-glassed shaped.

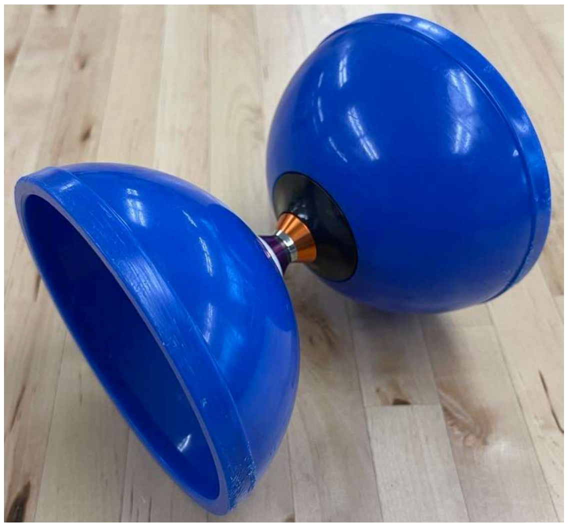


Figure 1. *A Blue Triple Bearing Diabolo.* This is the diabolo I used for my project. The orange and purple axle is the triple bearing that reduces the effects of friction when spinning.

Related Works

Two main research teams have worked with modeling diabolo movements. Von Drigalski et al. developed an analytical diabolo predictor that estimates the next state based on the current state and the stick positions using a constrained forward Euler method (2020). Murooka et al. created their algorithm, Diabolo-Manipulation-Net, and it's trained using the diabolo’s pitch and yaw, the robot's arm height difference, and the robot's spin speed (2020). This shows that it is possible to model diabolo orientation.

My research mainly focuses on developing von Drigalski et al.'s research further. Von Drigalski et al.'s analytical model of the diabolo predicts the coordinates of the diabolo and its rotational speed (2020). I developed an algorithm to supplement this by predicting the pitch of the diabolo as well. Rodriguez Ladron de Guevara et al. found a linear relationship between the offset of the diabolo performer's hands and the change in pitch of the diabolo (2018). I also determined that there is an inverse relationship between diabolo rotational speed and pitch angle.

My algorithm uses the forward Euler method to predict pitch angle, using the time derivative of pitch, approximated by the ratio between the offset of the sticks and the rotational speed of the diabolo multiplied by a constant, determined empirically. Essentially, the change in pitch is directly related to the offset and rotational speed ratio.

Additionally, von Drigalski et al. mentioned that a residual-physics learning neural network, R-PLNN, can improve the analytical model (2020). Therefore, I took the input required to predict the movements of a diabolo, current diabolo values, current stick values, and next stick value, and I trained a neural network using TensorFlow to predict the difference between the analytical values and the real values.

The paper that von Drigalski et al. referenced about residual physics is *TossingBot: Learning to Throw Arbitrary Objects With Residual Physics* by Zeng et al. (2020). They sought to create a model that can allow a robot arm to accurately throw random objects from a container to a target box. The article's Section IV pertains to my project. It explains how a part of the throwing model predicts a residual on top of the physics-based controller's ("ballistic equations of projectile motion") calculated velocity for the throwing speed (Zeng et al., 2020). The final velocity used is the sum of the residual and the calculated velocity. Zeng et al. state "this enables our models to leverage the advantages of physics-based controllers (e.g., generalization via analytical models), while still maintaining the capacity (via data-driven residual δ) to account for aerodynamic drag and offsets to the real-world projectile velocity (conditioned on the grasp), which are otherwise not analytically modeled" (2020). I used this principle in my project.

Methods

Gazebo and ROS

I used Gazebo and looked into ROS because von Drigalski et al. used Gazebo and ROS for their research (2020). Early on in my research, I reached out to Dr. von Drigalski and received access to their code repository. They later released the code on 10/14/2021 at https://github.com/omron-sinicx/diabolo.

There are three main notes I want to make about my experience using Gazebo and ROS.

My first note is on VNC servers. Originally, I was working in a Virtual Box Ubuntu virtual machine on my laptop. It was important to use Ubuntu because Gazebo only works on Ubuntu. The virtual machine was inefficient and slow to work with because Gazebo heavily relies on graphical processing. Lead Sysadmin, Lauren Delwiche, helped me set up a Sys Lab workstation for my project. I recommend that people don't try to use a VNC server to access a Sys Lab workstation for using Gazebo because the port forwarding takes time and effort and it's not as efficient as directly working on the PC. If it works, it's great to work with, but I would have had an easier time and accomplished more if I just did everything directly on that PC. Also, working on that PC directly would have helped me by making it easier to connect to Gazebo for my website, since it would have been on the same machine. But, this separation did save me in the end because a week before TJ Star the workstation failed due to a part frying.

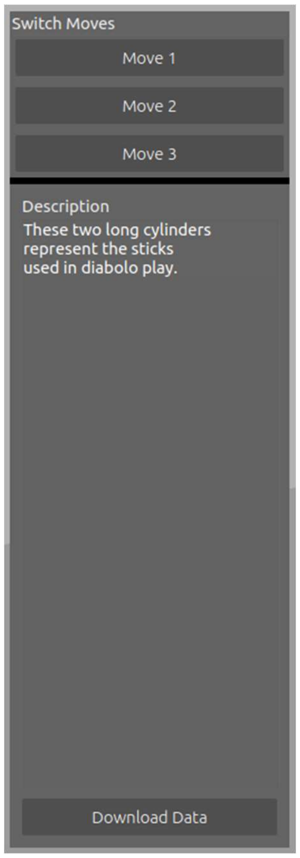
For reference, the connection to the workstation required proxy jumping because to connect to any Sys Lab computer, one needs to connect to TJ's remote access server, RAS, and then to a specific workstation or computer. The command I used to connect to Thompson so that I can access the VNC server is “ssh -v -L 5901:localhost:5902 -J 2022jcui@ras1.tjhsst.edu jcui@thompson”. The VNC server on Thompson is on localhost:5902 and the proxy jump connects my localhost:5901 to the localhost:5902 on Thompson. This allows me to use VNC viewer to connect to my localhost:5901, which is connected to Thompson's localhost:5902. Because of the several steps between my laptop's connection to Thompson, there were periods when there was longer latency or weaker connection, and the VNC server just didn't display anything. There was also another issue of not being able to view the VNC server and the actual desktop on Thompson at the same, but redownloading the display manager resolved this issue.

My second note is about the process of downloading and installing Gazebo and ROS. Throughout the process of setting up Gazebo and ROS, there were issues with downloading those packages and having them work properly. I had problems with different versions and Gazebo or a Gazebo server not loading. However, eventually, after installing a new OS and downloading the combined Gazebo and ROS software package, it worked. When downloading Gazebo and ROS, be patient and be prepared to try again. I used the steps stated by Gazebo on this page: https://classic.gazebosim.org/tutorials?tut=ros_installing&cat=connect_ros

My third note is on using von Drigalski et al.’s repository as ROS packages. I attempted to run their GitHub repository’s code. I set up an environment with the same software version as what they used: Ubuntu 18.04 with ROS Melodic. However, the project structure is not quite exactly what ROS packages should be structured according to ROS Melodic's documentation. ROS uses catkin workspaces to create packages. Though, this is based on my limited understanding of ROS, so I might have missed something. In the future, those who are more apt at ROS can look further into this. I faced several issues with the package structure, and I also tried to restructure the folders in accordance with catkin workspaces' requirements. I also tried installing the dependencies, but there were incompatibilities that I couldn't resolve using the information provided in the repository. I did come up with a solution of how I can use their algorithms: if I look at the specific parts of the repository that performs diabolo motion prediction, isolate it, and make it into an independent program I can run. I decided to create the program and convert the algorithm from C++ into Python since that is more familiar to me. There are more details in the section Modeling and Prediction under Analytical Model.

GUI interface: Gazebo GUI

I created a GUI for displaying Diabolo moves using Gazebo's API and Qt's C++ GUI library. As shown in Figure 2, the GUI is an overlay on top of Gazebo and the buttons can prompt Gazebo to display different moves on the diabolo and stick models on the screen. The diabolo model is represented by two circles, one for each side of a diabolo. For each move clicked, it also displays the description for that move. At the bottom of the overlay, there's an option to download the data. However, once I got GzWeb and GzWeb integration into a custom website working, I moved passed using a Gazebo GUI overlay, since a website is more accessible than a GUI that requires many steps of downloading software, setting up, and using the Linux terminal, which is not user friendly.



These two long cylinders
represent the sticks
used in diabolo play.

Figure 2. *Gazebo GUI Overlay.* There are three buttons for three example moves at the top. The middle section contains the description of the currently selected move. A button to download data is at the bottom.

The main resources I used to build the GUI include the Gazebo tutorials on the GUI plugin and the documentation on the Qt library's website. It is a little difficult to work with, since I was working with C++ for the first time, but it is not impossible to figure out because of the detailed documentation on Gazebo and QT's websites.

GUI interface: Gzweb

To set up the server, I used a Sys Lab workstation, set up by Lauren Delwiche. The OS is Ubuntu 20.04, and the software is Gazebo version 11.9.0 and ROS Noetic. Installing GZweb is relatively easy: just follow the instructions on the website. Gzweb needs Gazebo or gzserver (Gazebo server without the graphical interface) running as well. It's important to make sure that before running the command “npm run deploy” (i.e. building models) the system’s main Python version is Python 2 because that is what GzWeb uses for deploying. Figure 3 shows the GzWeb user interface that Gazebo provides natively.

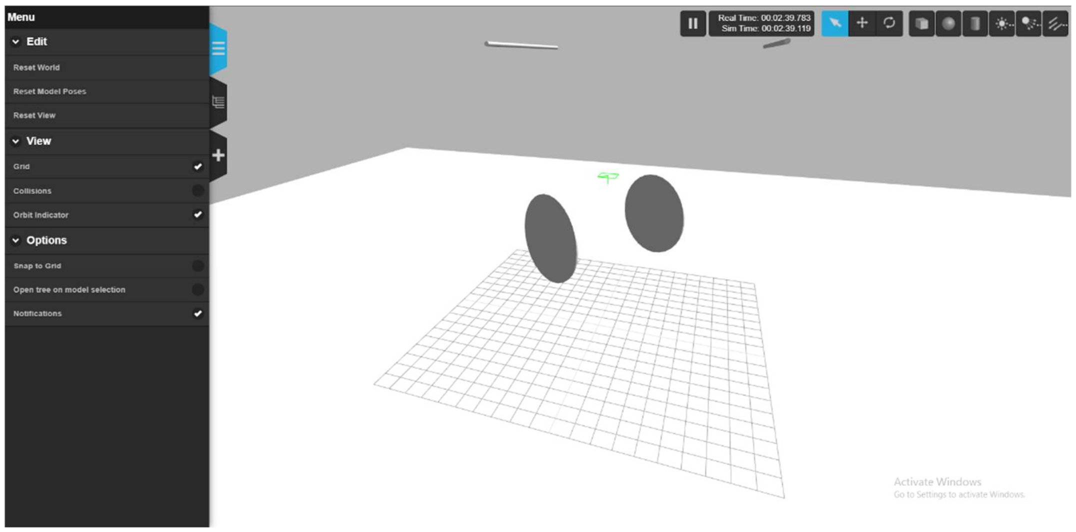


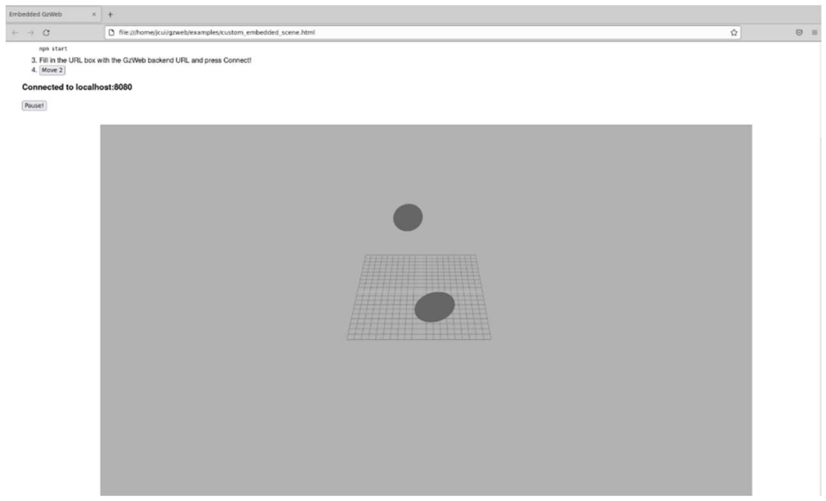
Figure 3. *The GzWeb User Interface.* This is the user interface provided by GzWeb. The left side panel is the menu, and the control options are at the top, right corner.

A problem I faced is that GzWeb doesn't support Gazebo GUI overlay, so that's why I had to find a method of adding my interface to GzWeb. To have a custom GUI to control the models, I needed to know what commands GzWeb uses to manipulate the objects through the webserver. This is why I started reading and investigating the “gz” JavaScript files in the folder, “gz3d/src”.

In the file “gzgui.js”, there is a class called GZ3D.Gui that displays the scene, a representation of what Gazebo displays, and the class contains the functions for the interface controls. Buttons on the site provided by GzWeb can call those functions. The functions emit a "signal" using a globalEmitter object, and the emitter can detect those “signals”, like an action listener. Once a signal is detected, the emitter will run the corresponding function associated with that signal. Additionally, the scene representing the Gazebo space that contains the objects has many useful methods. This scene is a GZ3D.Scene object, defined in the file “gzscene.js”, and is passed into the GZ3D.Gui object for the GUI to use. I can call the scene object's methods to change the properties of the objects on the scene.

I realized that "spawning", adding an object to the scene, using code introduces many additional and unnecessary steps, since this program and object spawning is designed for human interaction and not code manipulation. However, a straightforward solution is to create a world, which can be thought of as a predefined scene, that has already spawned these objects, but out of sight, so that when the objects move to the center, they appear like they were spawned. This also runs faster than the previous attempt because it doesn't need to wait for the server to respond with a new object. Moving the objects is simple because I can get the object by name from the scene using getByName and I can change the fields of the object, like orientation or coordinate position.

Because of the complexity of the default GzWeb frontend, I decided that finding a way to integrate a GzWeb scene into my website would be the best course of action. The GzWeb repository has an example folder of different HTML files showing the different features. One of the features is connecting to the Gzweb webserver within another website so that the Gzweb webserver scene is contained within a window on the other website, as shown in Figure 4. It works well, since it integrates smoothly into the website I created and it's easy to add custom code within the code they already have.



```
npm start
```

3. Fill in the URL box with the GzWeb backend URL and press Connect!
4. Move 2

Connected to localhost:8080

Figure 4. *The "embedded_scene.html" Example.* The example site is connected to a running GzWeb instance, and that allows it to display the objects from Gazebo.

Gazebo and ROS Conclusion

Based on my experience, I recommend you to work with Gazebo and ROS only if you have beyond a rudimentary level of understanding of Gazebo and ROS or if you have to interface with robots that require Gazebo/ROS. Otherwise, if you plan to use Gazebo as I did, I recommend other avenues of graphical representations of 3D objects.

Motion Capture

I used TJ's Vicon Motion Capture system to gather diabolo performance data. The system has nine cameras, consisting of MX T20s and MX T40s, as shown in Figure 5, all connected to the MX Giganet, as shown in Figure 6, which provides power to the cameras and establishes a data connection between the cameras and the MX Host PC. The MX Host PC is simply a PC that runs the Vicon Tracker software, which operates the cameras and processes the data received from the cameras through the Giganet.

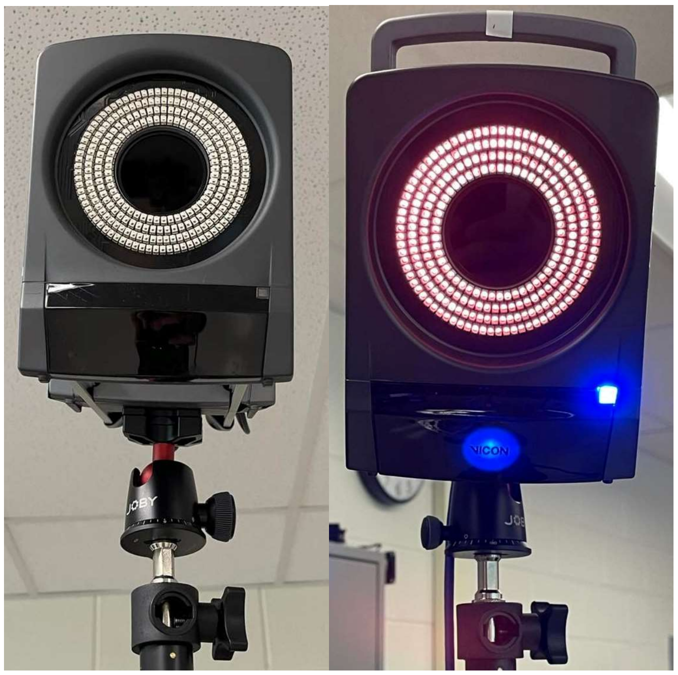


Figure 5. *Vicon MX Cameras.* These are TJ’s Vicon Motion Capture System’s MX cameras. On the left is an MX camera in the off state, and on the right is an MX camera turned on.

Figure 6. *The GigaNet.* This is a top-down view of the Giganet. The wires on the bottom right corner connect the MX cameras to the Giganet.

The Vicon Motion Capture system can capture objects' motion through the use of infrared light. The Vicon cameras beam out IR light and retroreflective markers, markers that specifically bounce light back in the direction it came from, fixed on objects will reflect back the IR light to the cameras' sensors. The cameras send that data to the Giganet and the Giganet sends the data to the Host PC for computer vision processing. The Tracker software combines the information from each of the cameras to determine specific locations of markers within the 3D space.

Using the Vicon Motion Capture System

Vicon's YouTube channel has an informative playlist on using their Vicon T-Series cameras and the Vicon Tracker software: https://www.youtube.com/watch?v=C8Nca7BDoK0&list=PLxtdgDam3USXD2G3DucxMBCZGM89cIdf-. This section has my additional insights into the process.

It's important to make sure the cameras are arranged facing the target region you want to track. I used the Vicon Tracker software to help me arrange the cameras: I placed markers on the ground and looked at where they show up in the camera view to align the cameras to the right positions and angles. Figure 7 illustrates the arrangement of the cameras within Mr. Kosek's room.

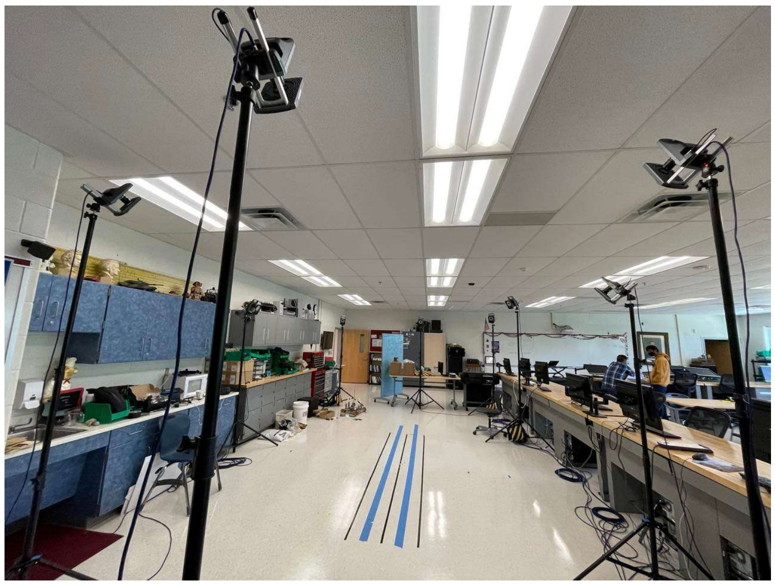


Figure 7. *Vicon Motion Capture System.* The MX cameras are arranged so that they face inwards and downwards towards the region used for motion capture.

I have found that turning on the GigaNet after opening up the Tracker software ensures that the Host PC and the GigaNet will make a connection. It's important to make sure that all the cameras are connected to the GigaNet, and the cameras will show a red ring once all of them are connected. The Tracker software under the "System" tab will show a play button symbol next to each camera's name once the camera is connected.

Before starting anything, it's important to have the appropriate settings. The relevant settings to my project are located under the properties of the "Local Vicon System"), and clicking "Show Advanced" provides more options. In my project, I changed the FPS and marker speed settings. As FPS increases, the capture rate increases, but the resolution will decrease due to hardware limitations. Increasing marker speed will increase the speed of markers the system will consider. The maximum value of marker speed is 10. The settings I used for recording are 600 fps, which is not the max to preserve enough resolution, and a marker speed of 10 to capture the rapid turning of the diabolo. It’s crucial to not change the IP address settings because that specifies the connection the Host PC makes with the Giganet.

Calibration. An important step when using the motion capture system is calibration. The “Calibrate” tab, as shown in Figure 8, helps execute this process, and this step informs the Tracker software on how to relate the data received from cameras with the physical world.

The first step is applying the camera masks. In most environments, like TJ’s motion capture system’s location, Mr. Kosek’s room, there are reflective surfaces that aren’t retroreflective markers and have to be there for different reasons, such as the floor. It’s important to inform the cameras that those surfaces shouldn’t be tracked. The “Create Camera Masks” section can automatically cover the extraneous reflective surfaces by creating a mask. The best way to create the masks is to first select the camera view, as shown in Figure 9, and select all the cameras under the “System” tab. The extraneous reflective surfaces will appear as stationary white blotches. After clicking start, blue regions will appear on the various camera views, representing the mask, and they will flicker. It’s important to click stop when all the white blotches are covered by blue regions. Make sure to remove all retroreflective markers from the cameras’ field of view before doing this step or else the Tracker software will not properly create a mask.

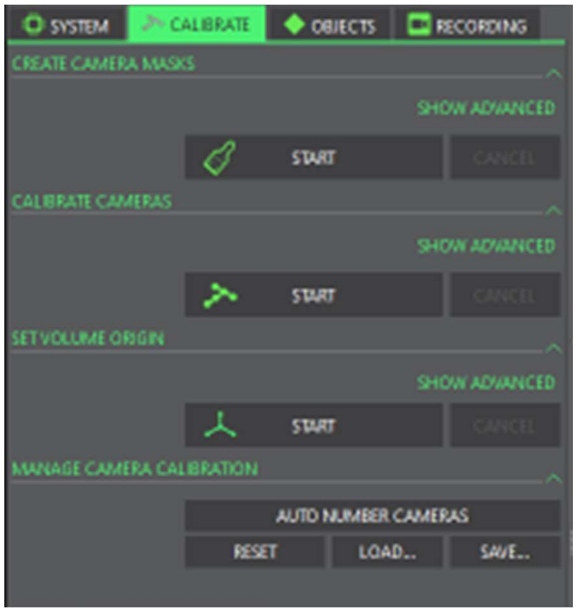


Figure 8. *The "Calibrate" Tab.* The "Calibrate" tab has four functions: create camera masks, calibrate cameras, set volume origin, and manage camera calibration.

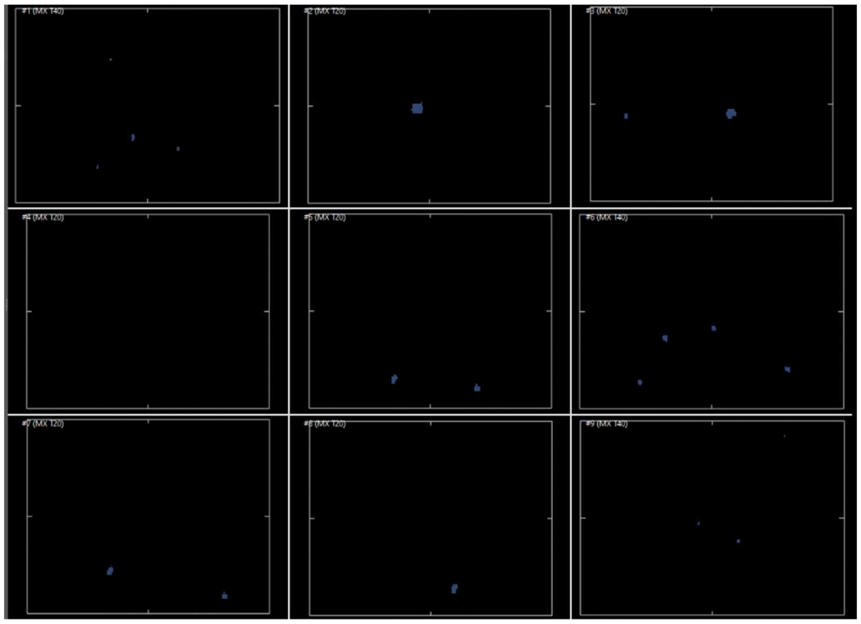
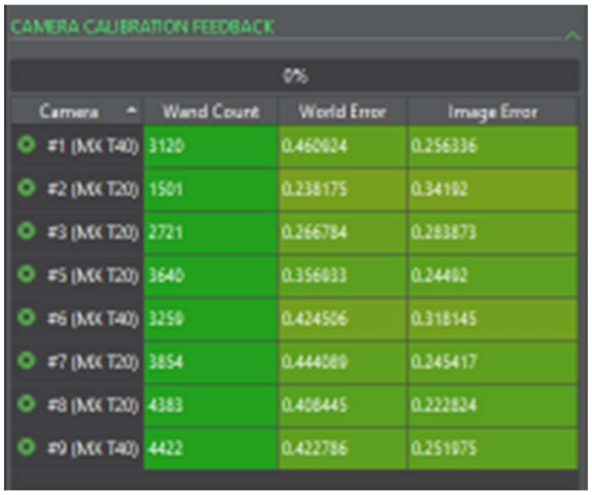


Figure 9. *Camera View.* This camera view shows in each part of this three-by-three grid what the nine MX cameras detect. The blue regions are the camera masks.

The second step is calibrating the cameras. This process utilizes markers on an object that the Vicon Tracker already knows the dimensions of; most of the time, it will be the provided Vicon Calibration Wand. Before starting the process, make sure the Camera view is still selected and all cameras are selected. After the process starts, I have to slowly wave the wand around the capturing space in front of the cameras, so that the Tracker software can coordinate the camera data with each other and reach a consensus on the collected wand locations. As the calibration process goes on, the Tracker Software will give feedback on the error for each camera so far, as shown in Figure 10. It's important to have high wand counts and low error values. The sidebar will show, for each camera, the color red when wand counts or error values are not adequate, and the color will gradually shift towards green as the system is more calibrated.


CAMERA CALIBRATION FEEDBACK

0%

Camera	Wand Count	World Error	Image Error
#1 (MX T40)	3120	0.460024	0.256336
#2 (MX T20)	1501	0.238175	0.34192
#3 (MX T20)	2721	0.266784	0.283873
#5 (MX T20)	3640	0.356033	0.24492
#6 (MX T40)	3259	0.424506	0.318145
#7 (MX T20)	3854	0.444089	0.245417
#8 (MX T20)	4383	0.408445	0.222824
#9 (MX T40)	4422	0.422786	0.251975

Figure 10. *Camera Calibration Feedback.* For each camera, during and after the calibration, this section, camera calibration feedback, will show the wand count captured and the error values.

As shown in Figure 11, in the cameras' views, there will be a rainbow trail of the wand's movement to show a visual representation of how many wand counts there are for each camera, and in the bottom, right corner, there's a color, from red to green, to show the extent the cameras are currently calibrated and will disappear once the camera is fully calibrated. A helpful feature for calibration: when looking at the actual cameras, on the bottom black bar, a light will turn off once that specific camera is calibrated. After all the cameras are calibrated, the Tracker software will show the final error values for each camera, and relatively good values should be in the range of around 0.2 to 0.4. These values might seem high, but in the message pane, it'll state the overall accuracy as a percentage, and it should be at least 99% if calibrated correctly.

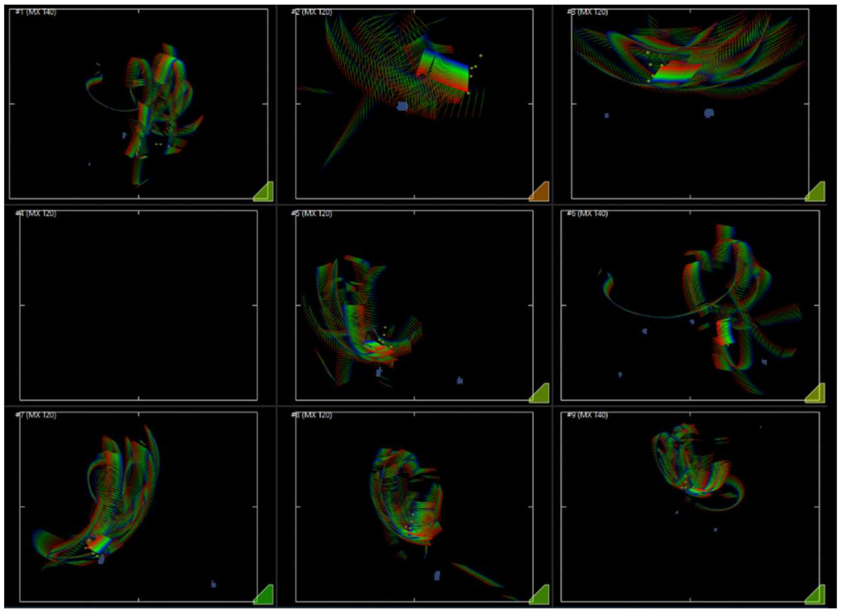


Figure 11. *Camera View during Calibration.* For each of the cameras, the camera view will show a rainbow trail for each wand position captured and the current calibration status at the bottom right corner.

An important consideration is that, even if the cameras were previously calibrated, before using the cameras, it's important to calibrate them again because any camera movements will introduce errors in all collected data. There will always be people and students who accidentally bump into the cameras.

The final step of calibration is to set the origin. This is best done in the 3D Perspective view, which shows the orientation of the cameras and the ground plane. Simply, just set the wand at a preferred origin location, click start under the “Set Volume Origin” section, and stop after the ground plane has shifted. The wand markers should be at the corresponding location in the virtual view as the physical, placed location.

Setting Up for Recording. In the “Objects” tab, markers are grouped together into objects. The objects allow you to track a position relative to the markers, usually the center.

As shown in Figure 12, my diabolo had three markers arranged as a non-isosceles right triangle, to reduce symmetry and increase the ability of the Tracker software to differentiate different orientations. The Tracker software object I created based on those markers is centered on the center of the diabolo instead of the makers themselves. As shown in Figure 13 and Figure 14, each of the two sticks had four markers, arranged in a kite shape and placed on a 3D printed holder based on the design von Drigalski et al. used (2020), the markers placements were intentionally placed differently between the left and right sticks to allow the Tracker software to better differentiate between the two. The object for the sticks in the Tracker software was centered at the tips of the sticks.

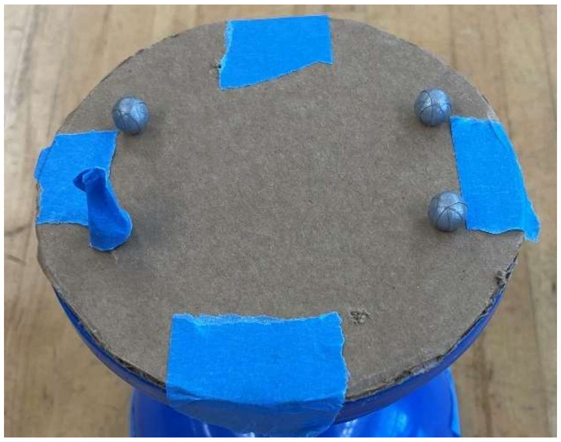


Figure 12. *Markers on the Diabolo.* This is the arrangement of retroreflective markers used for the diabolo. The retroreflective markers are the gray, spherical objects in the figure.

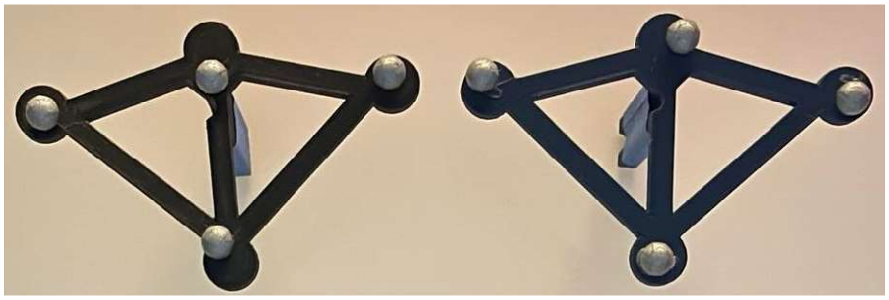


Figure 13. *Markers on the 3D Printed Holders.* These are the two 3D printed holders used to attach markers to the diabolo sticks.

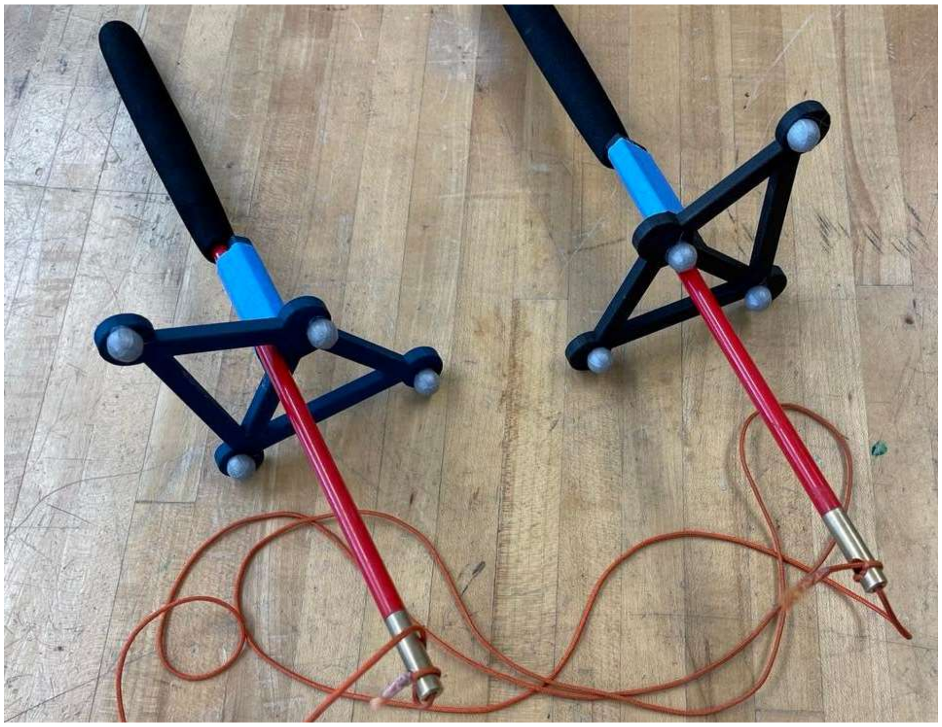
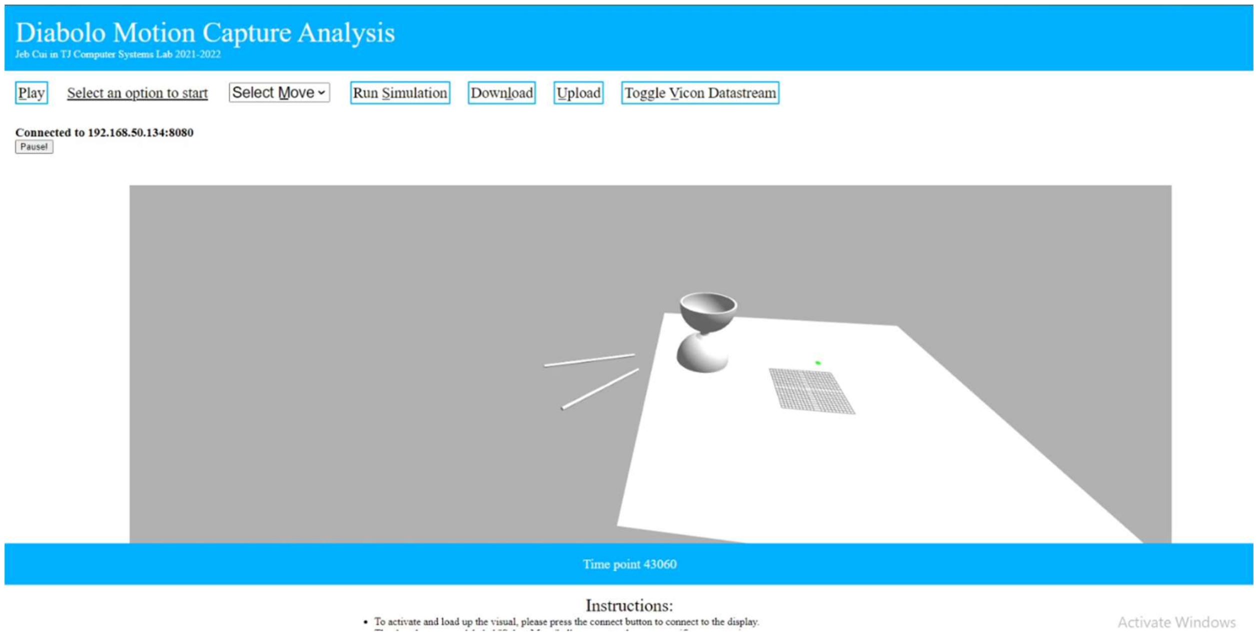


Figure 14. *Diabolo Sticks.* These are the diabolo sticks with the 3D printed holders, as shown in Figure 12, attached to them.

The “Recording” tab can record selected objects and replay specific recordings. The Tracker software saves the data as an X2D file, which is not useful for data analysis, but it can be converted to a CSV file. Additionally, under “Show Advanced”, it has the option to change how the orientations are formatted, either as Helical or Quaternion angles, and Quaternion angles were more useful for my project. Each of my recordings ranged from a minute to two minutes long.

Website

My website, as shown in Figure 15, has a Flask backend, written in Python. The Flask backend includes my models and Vicon Data streaming using SocketIO. The front end connects to Gazebo and displays movements from a recording. Additionally, there's also a pipeline setup for displaying the graphical representation of the model's output using Gazebo. The functionalities are developed mainly in JavaScript, as well as HTML and CSS.


Diabolo Motion Capture Analysis
Jeb Cui in TJ Computer Systems Lab 2021-2022
Play
Select an option to start
Select Move
Run Simulation
Download
Upload
Toggle Vicon Datastream
Connected to 192.168.50.134:8080
Pause!
Time point 43060
Instructions:
Activate Windows

- To activate and load up the visual, please press the connect button to connect to the display.

Figure 15. *My Website.* This is my website for this project, connected to a GzWeb server, showing the Gazebo objects representing the diabolo and sticks.

For the following features, displaying recorded motion capture data on my website and uploading data to run a prediction, the datasets are in CSV files, the structure of my datasets. I found a library called jQuery-csv that is an add-on to the jQuery library and allows me to process

CSV files into JSON objects. This allowed the website to process uploaded files and create JSON objects that JavaScript uses to store data. Using the uploaded data, I can change the objects, the diabolo and sticks' properties in the GzWeb scene to show the recorded movements of the data. For data used for running a prediction, the created JSON file can be easily passed through a POST request to the backend of my server to run a prediction.

The Vicon Tracker software has support for streaming data through a port when using the Vicon Datastream SDK. Vicon Datastream SDK has support on Windows as a Python library. I used their startup example to add this feature to the Flask backend. For streaming data, I needed to make sure that the machine running the Flask server has access to the internet port on the Host PC running the Vicon Tracker software. I used an ethernet connection for this. Additionally, I needed a way to stream the data from the backend to the frontend. I found SocketIO, which uses EventLet, and it allowed me to stream the data stream from my backend from the Tracker software onto the website.

I discovered the following later on in my project but it is more relevant to include here. There are several obstacles to hosting my site on a web server online. The Python version of the Vicon Datastream SDK is only supported on Windows. However, many hosting services, for example, TJ's director, is on Linux. Additionally, Director prohibits running machine learning on their servers, so I had to find an alternative, since my website uses machine learning. Additionally, SocketIO requires further steps to be supported on different hosting platforms. For example, Heroku doesn't support the implementation I wrote locally without further modifications that were beyond the timeframe I was working with.

Modeling and Prediction

Developing the actual model happened in two steps: the analytical model and the machine learning model.

Analytical Model

I transpiled von Drigalski et al.’s algorithm, located in the file “diabolo_play/src/diabolo_motion_generator.cpp” within their GitHub repository, into Python to remove the C++ specific library dependencies, make it integratable to a Flask web server, and connect it with Python machine learning libraries for the residual-physics learning neural network. I used the following libraries: vectormath, numpy, math, pyquaternion, pandas, and scipy. I wrote the following classes to implement the algorithm: DiaboloSimConfig, PoseArray, Pose, MyQuaternion (which extends pyquaternion), DiaboloState, Transform, and DiaboloPredictor.

Von Drigalski et al.'s analytical model does diabolo prediction in four steps (2020). First, the model predicts the rotational speed based on the previous rotational speed, how much of the string moved along the axle, and a friction coefficient. Second, using a forward Euler method, the model determines the position and velocity using the previous time step's values. Third, the model manipulates the diabolo state based on whether it's on the string normally, loosely on the string, or flying. Fourth, the new stick positions define an ellipsoid, with the sticks at the foci and the length of the major axis equal to the length of the string, and if the diabolo's predicted position is outside the ellipsoid, it's moved onto the closest point on the ellipsoid. The displacement is added to the diabolo velocity, representing the velocity added by the pull of the string. Figure 16 shows the diagram von Drigalski et al. used to show the process of step four.

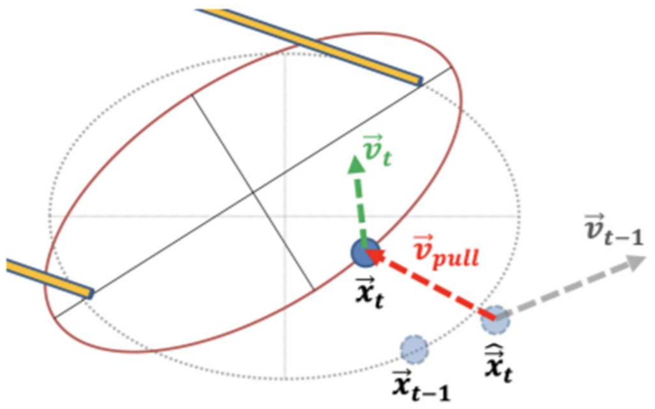
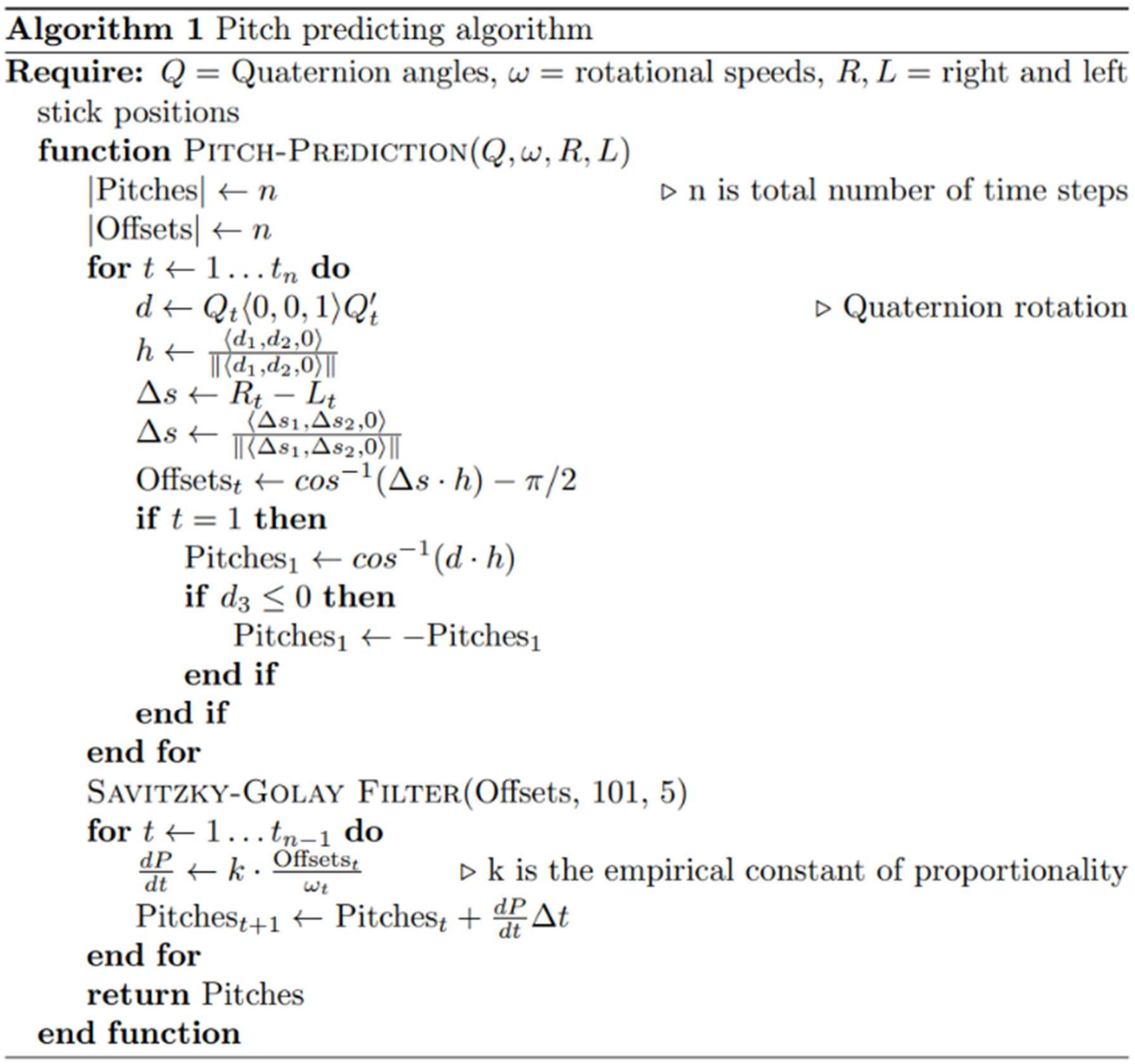


Figure 16. *Step Four of von Drigalski et al.'s Analytical model.* This diagram was created by von Drigalski et al. (2020). This shows how the diabolo position and velocity are updated using the ellipsoid defined by the sticks.

Pitch Prediction Addition. As mentioned in the related works, the change in pitch is linearly related to the offset of the performer's hands (Rodriguez Ladron de Guevara, 2018), and this is the same as the offset of the sticks. The full pitch predicting algorithm is located in Figure 17, and the result of running the algorithm on a section of data is in Figure 18.



Algorithm 1 Pitch predicting algorithm

```
Require: $Q$ = Quaternion angles, $\omega$ = rotational speeds, $R, L$ = right and left
  stick positions
  function PITCH-PREDICTION($Q, \omega, R, L$)
      $|\text{Pitches}| \leftarrow n$                              ▷ n is total number of time steps
      $|\text{Offsets}| \leftarrow n$
      for $t \leftarrow 1 \dots t_n$ do
          $d \leftarrow Q_t\langle 0, 0, 1\rangle Q'_t$                     ▷ Quaternion rotation
          $h \leftarrow \frac{\langle d_1, d_2, 0\rangle}{\|\langle d_1, d_2, 0\rangle\|}$
          $\Delta s \leftarrow R_t - L_t$
          $\Delta s \leftarrow \frac{\langle \Delta s_1, \Delta s_2, 0\rangle}{\|\langle \Delta s_1, \Delta s_2, 0\rangle\|}$
          $\text{Offsets}_t \leftarrow cos^{-1}(\Delta s \cdot h) - \pi/2$
          if $t = 1$ then
              $\text{Pitches}_1 \leftarrow cos^{-1}(d \cdot h)$
              if $d_3 \leq 0$ then
                  $\text{Pitches}_1 \leftarrow -\text{Pitches}_1$
              end if
          end if
      end for
      SAVITZKY-GOLAY FILTER(Offsets, 101, 5)
      for $t \leftarrow 1 \dots t_{n-1}$ do
          $\frac{dP}{dt} \leftarrow k \cdot \frac{\text{Offsets}_t}{\omega_t}$          ▷ k is the empirical constant of proportionality
          $\text{Pitches}_{t+1} \leftarrow \text{Pitches}_t + \frac{dP}{dt}\Delta t$
      end for
      return Pitches
  end function
```

Figure 17. *The Pitch Predicting Algorithm.* This shows the three parts of predicting pitch: calculating offset, applying the Savitzky-Golay filter, and doing a forward Euler method calculation.

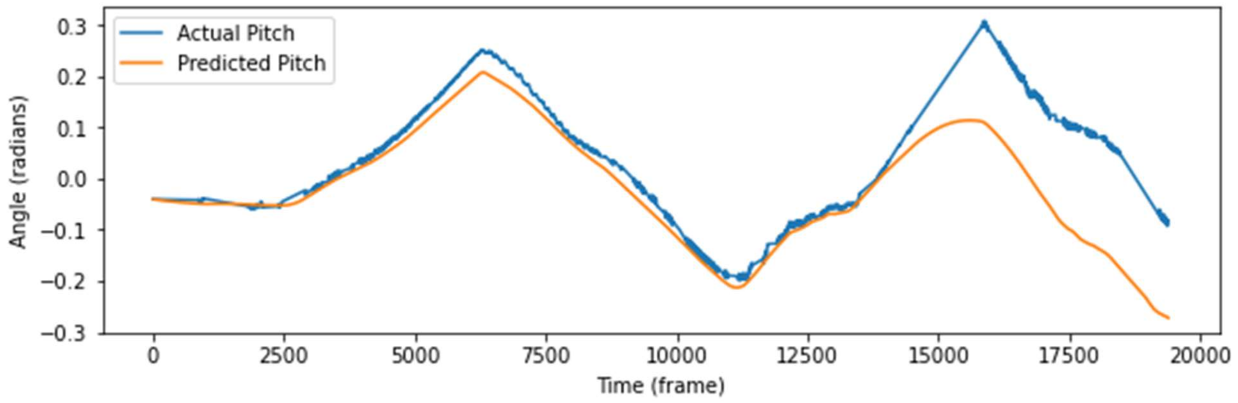


Figure 18. *Pitch Predicting Algorithm Result.* This figure is a graph of actual and predicted pitch values. The prediction diverages from the actual values at the end, starting at the sharp jump.

I defined the offset angle as the angle formed between the projection of the vector perpendicular to the axle onto the plane defined by the floor and the vector from the left stick tip to the right stick tip also projected to the plane of the floor. The offset angle is the angle θ in Figure 19 in the section Pitch Mechanics Derivation. The first vector is parallel to $\vec{r}$, and the second vector is parallel to $\vec{F}$. Then, I calculated the offset for the duration of the section of data I worked with. After getting those values, I use a Savitzky-Golay filter to smooth the offset angle data due to the noise and frame skips in data.

Based on mechanics derivations, I determined that the relationship among the change in pitch, offset angle, and angular speed is $\frac{d\alpha}{dt} \approx k\frac{\theta}{\omega}$ in which $\frac{d\alpha}{dt}$ is the change in pitch, k is a constant determined empirically, θ is the offset angle, and ω is the angular velocity. The next section, Pitch Mechanics Derivation, describes how the equation was determined.

The next step of the pitch predicting algorithm is to calculate the ratio of offset angle and angular velocity, as predicted by von Drigalski et al.'s algorithm (2020).

I use the forward Euler method to predict pitch angle based on the approximation of the pitch time derivative. The starting pitch is calculated as the angle between the vector defined by the axle and the plane defined by the floor. The time derivative of pitch is $\frac{d\alpha}{dt} \approx k\frac{\theta}{\omega}$. As stated before, this relationship is based on Rodriguez Ladron de Guevara et al.'s work (2018) and my mechanics derivations.

Pitch Mechanics Derivations. I made three important assumptions for my derivations. First, friction doesn't affect or minimally affect the effects of hand offset on the pitch. Second, the tension force on both sides of the diabolo is generally the same, which is true because the only time the tension force would change is when I'm tugging the diabolo to speed it up. This is also why I needed to filter out the noise from my values, since tugging the diabolo introduces a rapid change that the motion capture system may or may not detect. Third, I used common approximations for small values of x: $\tan(x) \approx x$ and $\sin(x) \approx x$.

As shown in Figure 19, the situation of these derivations is that the strings on the diabolo are offset by some amount and the horizontal tension force applied to the axel is $\vec{F}$. The offset is defined by the angle between the direction of the force $\vec{F}$ and the radius $\vec{r}$ from the center of the axle to the point where the force $\vec{F}$ acts on. Also, ω is the rotational velocity.

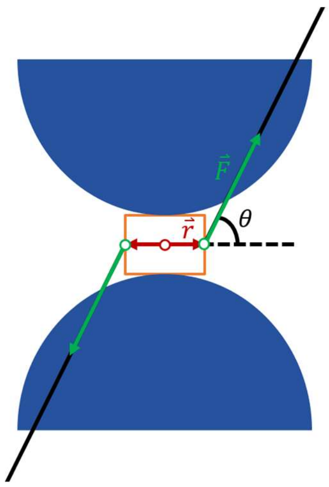


Figure 19. *Forces and Torque of the Diabolo.* This is a top down view of the diabolo with an offset to the sticks that results in the string going in two directions. The angle of offset is the same as θ, the angle between $\vec{r}$ and $\vec{F}$. The torque vector due to the offset is coming out of the page.

The torque caused by the offset of the strings is $\vec{\tau} = 2(\vec{r} \times \vec{F})$. The factor of two is due to both sides of the string contributing to the torque. The magnitude of the torque is $\tau = 2\mathrm{rFsin}\theta_t \approx 2\mathrm{rF}\theta_t$ where θ_t is the angle between $\vec{r}$ and $\vec{F}$ as shown in Figure 19 at time t.

Let $\vec{L}$ be the angular momentum of the diabolo. An important concept to remember is that torque is equal to a change in angular momentum over a change in time. Figure 20 shows the relationship between torque, over a certain amount of time, and angular momentum vectors. The angular momentum and torque, over a certain amount of time, form a right angle. This torque can change angular momentum, as shown by this equation: $\vec{L}_{t+1} = \vec{L}_t + \Delta t \cdot \vec{\tau}$, in which Δt is the difference in time between $t + 1$ and t.

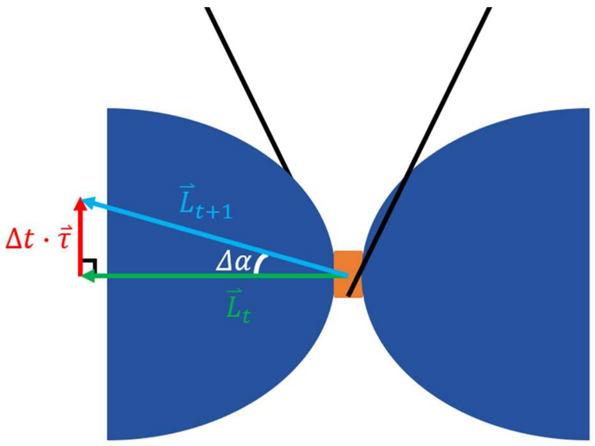


Figure 20. *Angular Momentum of the Diabolo.* This is a profile view of the diabolo. If a person is standing to the left and viewing the diabolo, it will be spinning counterclockwise. On the closer side, the string will be to the right, and on the farther side, the string will be to the left.

As shown in Figure 20, $\vec{L}_{t+1}$, $\vec{\mathrm{L}}_t$, and $\Delta\mathrm{t} \cdot \vec{\tau}$ form a right triangle. Note that this triangle does not have to be orientated above the horizontal, it can also be below. The angle between the new angular momentum and the old angular momentum, $\Delta\alpha$, or the change of the diabolo's pitch is calculated by the following: $\Delta\alpha = \tan\left(\frac{\Delta t\tau}{L_t}\right) \approx \tan\left(\frac{2\Delta t}{L_t}\right) \approx \frac{2\Delta tr}{L_t} = \frac{2\Delta trF}{I_D\omega_t}$ where I_D is the moment of inertia of the diabolo around the axle. Let $\mathrm{k} = \frac{2rF}{I_D}$, the constants of this equation. Then, $\Delta\alpha \approx \mathrm{k}\frac{\theta}{\omega}\Delta\mathrm{t}$.

If I divide both sides by Δt, I then have an approximation for the rate of change of α, the pitch: $\frac{d\alpha}{dt} \approx \frac{\Delta\alpha}{\Delta t} = \mathrm{k}\frac{\theta}{\omega}$.

If Δt is small enough, it will be accurate enough to do a forward Euler method approximation using the time derivative of the pitch. I didn't calculate k analytically because it's not just the values within the equation that k would wrap together when applied to physical values, but also friction, deviations in the moment of inertia, and imperfect tension balance. Additionally, getting an empirical value of k would be much more applicable to the physical world. I calculated k by taking the change of pitch values and the offset and angular velocity ratio of many time points and calculating k_t for each of those times and taking the median of the values. My empirical k value is approximately 0.00186.

Residual physics Learning Neural Network

As shown by Zeng et al., developing a model to predict a residual that's added to an analytical model's predictions for the final prediction can improve the performance of the prediction overall (2018). I trained a neural network to predict the difference between the analytical model's diabolo position and orientation prediction and the actual values, and this model is called a residual-physics learning neural network, or R-PLNN. The values from both the R-PLNN and the analytical model contribute to the predicted values for the next time step as shown in Figure 21.

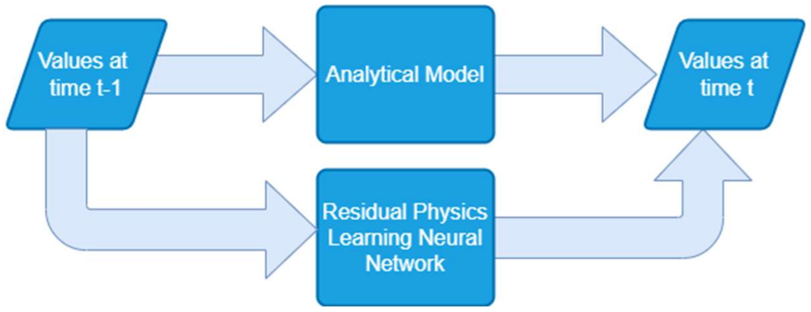


Figure 21. *Model Flow.* This shows the analytical model and the R-PLNN takes in values and their predictions combined together to determine the values at the next time step.

I used the Keras API in the TensorFlow library to develop the R-PLNN and SciKit Learn for preprocessing and creating the training dataset. To create the training dataset, I ran the prediction using my implementation of von Drigalski et al's algorithm. The inputs are the current diabolo position, velocity, and rotational velocity, and the current and next stick positions. My training datasets' input values are calculated similarly to how von Drigalski et al. determined their values in their neural network (2020). The velocity is calculated using the current position and the next position of the diabolo. The rotational velocity is generated from running the analytical model.

Then, I used rotational velocity and stick data to predict pitch. Afterward, I prepared the training data so that I'm predicting the difference between the diabolo position and pitch compared to the real data. This is calculated by subtracting the predicted from the real values. The output values are the diabolo position and pitch difference for the next time step.

The R-PLNN is a Keras sequential model with three hidden layers of 100 nodes using the ReLu activation function, like the neural network von Drigalski et al. created (2020). I trained the model over 300 epochs using a batch size of 100.

An assumption I made to develop this R-PLNN is that the current values of the diabolo and the current time and next time of the sticks are related to the error of the analytical model's prediction for the next time step.

Results

The website contains these main functions: displaying data, running predictions using my model, and streaming Vicon motion capture data. Users can upload their data to simply display the movements or to run a prediction on them.

My model has two substantial additions to von Drigalski et al's diabolo predictor: diabolo pitch prediction and the use of a residual-physics learning neural network. The pitch predictor independently closely predicts the actual pitch of the diabolo. The R-PLNN has a mean squared error of 0.1321 m^2 for the main section of the recordings I tested. However, my model is slow on the scale of minutes because of the time the analytical model, pitch calculation, and R-PLNN takes to process the large datasets.

The analytical model with the help of R-PLNN resulted in an MSE of 0.188 m^2. The analytical model resulted in an MSE of 0.771 m^2.

Discussion and Conclusion

I was able to extend the prediction ability of von Drigalski et al.'s algorithm to include pitch, another aspect of the diabolo's orientation. The residual physics learning neural network was able to improve the performance of the analytical model, but the ability of it to improve the analytical model's predictions has to be weighed against the time it takes for it to predict the values. However, it still would be beneficial to use the combination of the analytical model and R-PLNN to do diabolo prediction because of the significant improvements.

A significant obstacle I faced is that it was difficult to get the motion capture system to continuously record, without dropping frames, the faster-moving diabolo, even after tweaking the settings, due to limitations of the hardware and coordinating camera arrangements with other groups using the system, who have different goals than I do. The site I developed allows users to interact with the research I've done, but it's not as approachable as I would have hoped because there are a fair number of steps required when using the features on my site. However, I believed that I did the best I could with the project.

Future Work

The research I did can be improved in the future in these potential ways. Improving the efficiency of the analytical model, such as ways to optimize the size of timesteps automatically; developing prediction for yaw, since a limitation to my testing was that even with my best efforts I wasn't able to completely balance the markers and it introduced some yaw not inherently caused by my stick movements; doing more testing on various machine learning model schemes to see which one might work better, such as different hidden layer sizes and counts; and experimenting on fixed axle diabolos, since I only used a bearing axle diabolo for my research.

References

Murooka, T., Okada, K., & Inaba, M. (2020, October 25). Diabolo Orientation Stabilization by Learning Predictive Model for Unstable Unknown-Dynamics Juggling Manipulation. IEEE/RSJ. International Conference on Intelligent Robots and Systems, Las Vegas, NV, USA (Virtual). http://ras.papercept.net/images/temp/IROS/files/3245.pdf

Rodriguez Ladron de Guevara, M., Daly, A., & Bajaj, S. (2018, February 19). Diabolo Skill Analysis – Human-Machine Virtuosity. Cmu.edu. https://courses.ideate.cmu.edu/16-455/s2018/501/diabolo-skill-analysis/

von Drigalski, F., Joshi, D., Murooka, T., Tanaka, K., Hamaya, M., & Ijiri, Y. (2020). An analytical diabolo model for robotic learning and control. ArXiv.org. https://doi.org/10.48550/arXiv.2011.09068

Zeng, A., Song, S., Lee, J., Rodriguez, A., & Funkhouser, T. (2020). TossingBot: Learning to Throw Arbitrary Objects with Residual Physics. ArXiv.org. https://doi.org/10.48550/arXiv.1903.11239