KUKA iiwa Fruit Ninja

Billal Iqbal, Michael Lu, Ryan Xiao
Department of Electrical Engineering and Computer Science (EECS)
Massachusetts Institute of Technology (MIT)
Cambridge, MA USA

***Abstract*—The premise of Fruit Ninja, an incredibly popular mobile game, is that the player must cut randomly generated fruits flying across the screen with a ballistic trajectory. For humans, the ability to track fruits, estimate their future position and create a plan on how to cut the fruits is easy. This project aims to incorporate the game into a robotic simulation to evaluate the performance of a Kuka iiwa robotic arm. By varying the trajectories and velocities of launched fruit, the robustness of the robots ability to track and slice fruits can be evaluated. The project experienced some technical difficulties with cutting multiple fruit simultaneously, modelling cutting dynamics, perception systems, and trajectory optimisation, which are discussed further in this report. Also, please find our code here: https://github.com/michaellu2019/kuka-iiwa-fruit-ninja.**

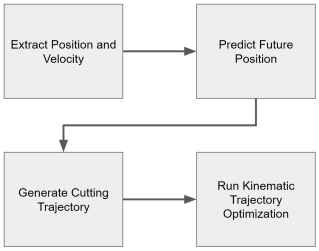

Figure 1. High-Level Robot Approach for KUKA iiwa Fruit Ninja.

I. Introduction

As an icon of modern society, Fruit Ninja has become a household name among any person that has used a touch-screen device. Almost everyone has either played before or knows how the game is played. Fruit Ninja is a mobile video game released in 2010 by Halfbrick where fruits fly up onto your screen and the goal is to slice the fruits by swiping through them with. The popularity has since expanded beyond the mobile game into the physical world, from fruit ninja in virtual reality to content creators slicing flying fruits with actual blades. So, if humans can do it, robots can too! Not only is making a robot to play fruit ninja fun, it also presents some interesting technical challenges.

The basic approach to creating a successful fruit ninja robot starts with perception with cameras to pinpoint the location of fruits and calculate their velocities. This data is then fed to the robot to calculate a possible point of intersection with the fruit and potential time of intersection. This is then used to calculate the most optimal swing trajectory and when to execute that swing. Hence, the goal will be to build a simulation that has fruits that generate and fly toward the robotic arm. The arm will hold a katana, determine positions and velocities of fruits flying in the air, and successfully intercept them in the air.

II. Related Work

The research work on robots wielding giant blades to perform tasks is admittedly sparse, likely for safety reasons, though there is some work on similar concepts that came up in our project. For example, the concept of cutting using robots has been documented in some papers such as ”Robotic Cutting: Mechanics and Control of Knife Motion”. However, this paper delves into the use of robots within kitchens and proposes strategies on how to press push and slice food whilst it is placed on a chopping board in a manner conducive to cooking. The forces and dynamics involved with cutting a stationary fruit resting on a smooth surface vastly differ from the conditions presented within this project.

Another paper related to the project is ”Catching Objects in Flight”. This paper details the ability to track objects launched towards a robot which is helpful for the project. However, a significant portion of the paper delves into the machine learning approach used to estimate properties such as the mass distribution, shape and material of the items launched at the robot to determine the optimal capture strategy.

Therefore, a rough baseline of knowledge could be taken from existing related work, whilst most of the work done throughout the project is unique to the requirements of the simulation. For example, there we no sources available on how to determine the energy required to cut the fruit, the deflection of the katana as it slices the fruits and so estimations and first order approximations were used for this project where necessary.

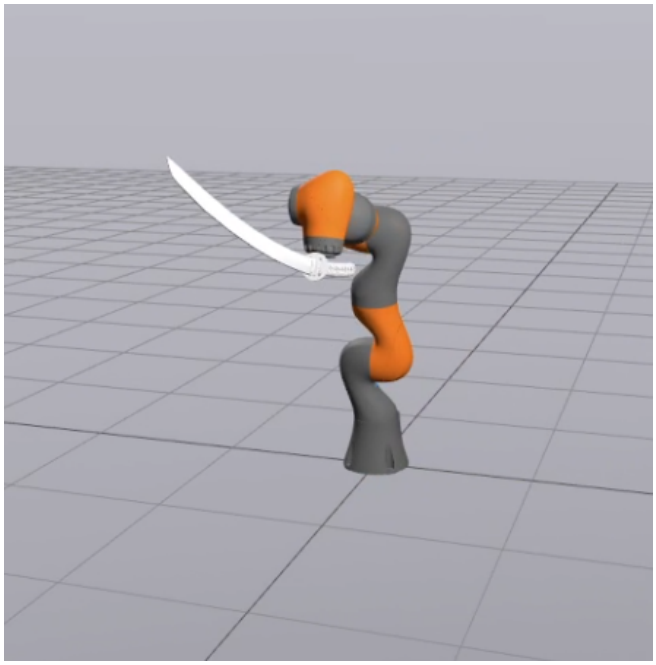

Figure 2. KUKA iiwa robot arm in Drake simulator with katana welded to final linkage.

III. Methods

III.1. Simulation Setup

We used PyDrake as the core of our project due to its ability to create high-fidelity physics simulations, its integration of off-the-shelf robot arm models and controllers, and multitude of robotics optimization solvers and controllers. We used a KUKA iiwa LWR robot arm for our robot of choice, which we welded a katana SDF to the final joint as the end-effector.

Fruits in the system were modeled as two half spheres mirrored and traveling together to appear as a single sphere. Spheres were the most optimal shape because they appear as the same shape independent of viewing angle. This vastly simplified the perception process that we use to determine the fruit velocity and position, explained in more detail in the Perception section. Half spheres were specifically chosen to model fruit slicing, which will be discussed in more detail later as well. The half spheres were generated by downloading a CAD model as an OBJ file and using it to generate an SDF file, which was be directly uploaded to a Drake scenario.

One important Drake-specific design we had to implement was using two multibody plants in the simulation. Because we had many free-floating fruit bodies in our scenario, we could not run trajectory optimization on the plant with all the elements, as it would consider the ”joints” of all fruits as decision variables, leading to a virtually unsolvable optimization problem. In addition, for collisions to occur easily, we needed bodies to be in the same plant. Thus, we

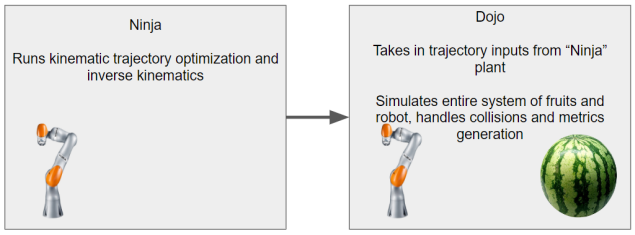

Figure 3. An interesting architectural choice we had to do to enable our optimization solver to work as well as our collision detection.

ended up creating two plants, one called the ”ninja,” which only had a KUKA iiwa robot arm and welded katana, and another plant called the ”dojo,” which had the KUKA iiwa robot arm and all the generated fruits. The ”ninja” plant’s solved trajectories were then connected to the input port of the ”dojo,” so that solved trajectories could be executed in the global plant system, allowing collisions between the robot katana and fruits to work.

III.2. Fruit Generation

In order to guarantee that the fruit is within reach of the robotic arm, a point, (x_d, y_d, z_d), is first randomly chosen within a certain distance in each direction from the katana that the arm holds. Then, an initial point, (x_0, y_0, z_0), is chosen at random from any point below the arm. This is done to emulate classic gameplay where fruits are shot up into the screen to be sliced. The initial z velocity of the fruit, v_{z0}, is also chosen at random, with the condition that it needs to be large enough to reach the height of the robotic arm. This can be guaranteed by adding this constraint:

$$v_{z0}^2 - 2g(z_0 - z_d) > 0 \Rightarrow v_{z0} > \sqrt{2g(z_0 - z_d)}$$

Because Drake contains a physics engine that runs in its simulations, algebraic kinematic equations can be used to solve for the initial x and y velocities, v_{x0} and v_{y0}, so that the fruit can travel from the randomly selected initial point to the randomly selected destination point. Using the randomly generated initial z velocity, v_{z0}, the time it takes for the fruit to get to the destination point can be solved with the following, $g = -9.8m/s$:

$$z_0 + v_{z0}t + \frac{1}{2}gt^2 = z_d \Rightarrow \frac{1}{2}gt^2 + v_{z0}t + (z_0 - z_d) = 0$$

And using the quadratic formula:

$$t = \frac{-v_{z0} + \sqrt{v_{z0}^2 - 2g(z_0 - z_d)}}{g}$$

The greater of the solutions is used to give the robotic arm more time to calculate its optimal swing path. From here, t can be plugged into these equations to obtain the initial x and y velocities:

$$x_d = x_0 + v_{x0}t \Rightarrow v_{x0} = \frac{x_d - x_0}{t}$$

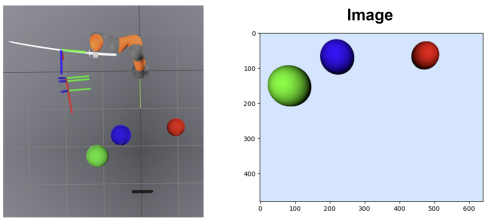

Figure 4. Top-view of the simulation set up with the extracted point cloud from the Intel RealSense D415 camera.

$$y_d = y_0 + v_{y0}t \Rightarrow v_{y0} = \frac{y_d - y_0}{t}$$

These initial conditions can then be directly set for the fruit objects in the Drake simulation.

III.3. Fruit Cutting

As briefly mentioned before, there is currently nothing in Drake that supports the slicing of objects in simulation as all objects are rigid bodies. So, half spheres were used to create this fruit slicing effect. These half spheres have a position defined at the center of their circular surfaces, so when 2 half spheres take on the same position and one is mirrored from the other, they reassemble a whole sphere. By setting equal positions and velocities for the 2 associated half spheres, they will always occupy the same position and take the form of a sphere when unhindered by outside objects. Surprisingly, they do not collide with each other when moving together so their trajectory does not alter.

Of course, the objective is to hinder these spheres by colliding with them with a katana. Drake has a handy tool that lets us check for collisions between any two objects in the simulation at a given time step. In order to utilize this, the simulation has to be advanced in sufficiently small increments so as to not skip over the point in time where the collision occurs. In this case, a time step of $.001$ seconds was sufficient to detect the collision between the katana and the fruit. At each time, all collisions between object pairs is checked, with each object identified by a unique object id. If the katana is in collision with either one of the half spheres of the fruit, then the robot has successfully intercepted and sliced the fruit. To emulate the randomness of the slice, the x and y velocity of one half sphere is randomly set to a value $\in [-1, 1]$. The other half's x and y velocities are set to the negative of the first half's x and y velocities so that it travels in the opposing direction. The result is a satisfying split of the fruit as the robot takes a slash.

III.4. Perception

For the perception system, an Intel Realsense D415 camera was used to provide a top view of the environment. Initially, there was a secondary camera to provide a side view, but with some simplifications the need for this camera was removed.

The perception system works by utilising the fact that all of the fruits being thrown at the robot at a given time are all different colours. The algorithm first extracts the depth and RGB images from the camera and generates a point cloud using this data. The point cloud is then filtered to remove points with infinite depth from the camera (points where there are no items in front of the camera) and then this point cloud is repeatedly filtered to extract points with a specific colour. By extracting points of a specific colour, a sub-set of points corresponding to one fruit are found. This essentially segments the image and identifies the different fruits since there is a 1:1 mapping between fruits and colours. Given the coordinates of all of the points that constitute a fruit, the fruit's centroid can then be found. The position of the fruit can then be compared between the current and previously recorded image to extract its velocity vector.

III.5. Fruit Segmentation

To segment the fruits, simply comparing the colours of points in the point cloud is not sufficient. Due to lighting conditions and the position of the fruit relative to the camera, the points along the fruit will appear to vary in colour.

Therefore, a thresholding approach is used. By allowing a range of RGB values to represent a specific fruit's colour, the perception system becomes much more robust and is less likely to only track a small subset of the actual fruit. Within the program each fruit has its own threshold values and each channel of the fruits colour can have its own threshold, allowing for high levels of control and accuracy.

The threshold works by providing an upper and lower bound for the specific value of a point's colour to be associated with a particular fruit colour. For example, if the fruits colour is [100,100,100] (RGB values) and the threshold is [10,15,20] then for the red channel, pixels can have a value between 90 and 110. For the green channel, the pixels can have a value between 85 and 115 and the same logic is applied to the green channel.

The addition of a light source could also be used to decrease dark regions on the surface of the fruits. However, this could result in glare off the surface of the fruits and care must be taken in determining the position, type of lighting and its intensity.

This approach is effective for the game since it enables an arbitrary number of fruits to be launched simultaneously, so long as arrays for the colours and thresholds for each fruit are specified. Whilst a contour based solution would work and allow for more representative fruit shapes, contouring and segmentation would become more computationally expensive and a vast number of fruits can be well approximated as spheres and so it is unnecessary.

III.6. Centroid Calculation

The next step of the algorithm is to compute the centroid of the identified fruits. By taking the mean of the x ordinates of all of the points associated with a particular fruit, the x ordinate of the centroid can be determined. This can be repeated to find the y ordinate of the centroid. For the Z ordinate, this is not applicable. Since our top view camera

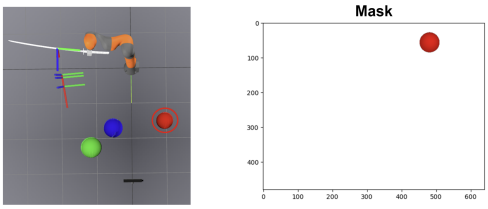

Figure 5. Fruit segmentation performed on the red fruit.

gives us information on how the fruit moves in the x (left or right of the robot) and y (forwards or behind the robot) axes, we would typically want the side camera to get information on how the fruit moves in the y and z axes. However, by simplifying the geometry of the fruits used to spheres this is not necessary.

The camera provides a projected image of the sphere, which is a circle. Along the circumference of the circle are the points with the largest depth from the camera. The points, due to the symmetry of a sphere are at the same height as the center of the sphere. Therefore, by taking the maximum depth of the points associated with a fruit, the z ordinate of the centroid can be found without a secondary camera and averaging.

The Drake library has built in functionality that automatically converts point cloud coordinates to be expressed in the world frame when the pose of the camera relative to the world frame is known. However, if this is not known then the centroid has to be expressed relative to the world frame so that the algorithm planning the robot's slicing can interpret the fruits position correctly.

III.7. Velocity Calculation

To calculate the velocity of the fruit, the difference in the centroid of the fruit between two images can be used. For the Fruit Ninja program created, collision physics between fruits was neglected and there are no external forces acting on the fruit, except gravity and the initial launching force. Therefore, tracking the fruit's trajectory is a simple kinematics problem where the position of the fruit and its velocity at that point are the only pieces of information needed to fully determine its trajectory. However, if fruits were allowed to collide and change trajectory, or there were external forces e.g. wind acting on the fruits, a more continuous tracking of the fruits would be required. To enable this functionality, the program takes images with a regular rate and continuously updates the velocity and position of the fruits. To do this a custom 'ImageProcessing' system is created. This inherits from the built in Drake LeafSystem and uses the 'DeclarePeriodicUnrestrictedUpdateEvent' function to take an image every 0.5 seconds. This system's input ports connect to the output ports of the camera, allowing it to take in image data. The output ports of this system connect to the input ports of a trajectory system which updates the expected trajectory of the fruit which can be used to plan the slicing motion of the robotic arm.

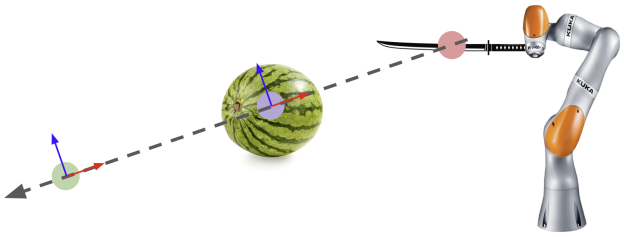

Figure 6. Slicing algorithm diagram. Given the expected position of the fruit at time t, our algorithm generated two additional waypoints that would form a line from the current robot's end effector pose to and through the fruit to an extrapolated end position.

III.8. Slicing Algorithm

Given the predicted position of the fruit, our next step was to more concretely define the steps that the KUKA iiwa needed to take to cut the fruit. We settled on generating a list τ of critical way points in a simple trajectory that the robot could follow. This started off by using the current position of the robot p_0 and the expected position of the fruit p and generating a vector v between the two points and normalizing it along with the same vector v but with $v_x = 0$:

$$v = p - p_0$$

$$v_{flat} = \frac{1}{\sqrt{v_y^2 + v_z^2}}[0, v_y, v_z]^T$$

v_{flat} was crucial for generating τ where the x-value stayed constant otherwise it would encourage the robot arm to over-extend itself when it could just rely on the length of the katana to reach fruits that were far out. Along with a relaxation of the position constraint in our solvers, this led to quick, robust optimizations that leveraged the length of the katana to extend the "work space" of the end-effector. From there, roll, pitch, and yaw angles were calculated using the "flat" slope.

$$\phi = \begin{cases} 0 & p_{0,x} > p_x \\ \pi & p_{0,x} \leq p_x \end{cases}$$

$$\theta = \arctan\left(\frac{-v_{flat,z}}{\sqrt{v_{flat,x}^2 + v_{flat,y}^2}}\right)$$

$$\psi = \arctan\left(\frac{v_{flat,y}}{v_{flat,x}}\right)$$

These values were then converted into a rotation matrix R that was applied as an orientation constraint to the cut point and end point of the line. These constraints would ensure the blade would cut the fruit perpendicular to its surface in order to slice it in half. Once this rotation matrix was created, p_0 was added to τ, then p was added to τ with the orientation constraint. Finally, a third point $p_f = p + \frac{v_f lat}{4}$ was added to τ with the same orientation constraint.

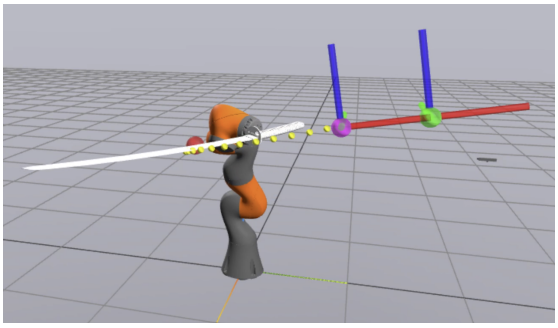

Figure 7. Visualized generated trajectory from slicing algorithm for one fruit. Red and green points represent start and end points, respectively, and purple represents the "cutting point," where the robot estimates it will intersect with the fruit to cut it. The yellow points show part of the blade path planned by kinematic trajectory optimization.

These three points would form the critical waypoints for the trajectory optimization methods discussed in the next section.

III.9. Trajectory Optimization

After crucial waypoints are determined by our slicing algorithm, the robot must then convert those waypoints into a joint-space trajectory to move the blade through those points in a certain amount of time such that the blade intersects with the falling fruit. We investigated two primary methods of generating this trajectory: inverse kinematics and kinematic trajectory optimization.

The points generated by the slicing algorithm are classified into three types of poses: start point, end point, and cutting points, which can be denoted as X^{start}, X^{end}, and finally X_i^{cut}, which can further decomposed into p_i^{cut} and R_i^{cut} to represent position and orientation of the i-th cutting point. From there, we formulate an optimization problem with these points with the necessary position and orientation constraints.

Starting with inverse kinematics, our optimization problem was as follows:

$$\begin{aligned} \min_{q} \quad & |q - q_0|^2 \\ \text{s.t.} \quad & X^{start} = f_{kin}(q) \\ & X^{end} = f_{kin}(q) \\ & \forall i, X_i^{cut} - \epsilon \leq f_{kin}(q) \leq X_i^{cut} + \epsilon \end{aligned} \tag{1}$$

where $q_0 = [0.0, 0.6, 0.0, -1.75, 0.0, 1.0, 0.0]^T$ and ϵ represented the bounds of accepted solved pose values, which were for us $[0, 0, 0.005]^T$ for the position and $\frac{\pi}{26}$ for the orientation. Our inverse kinematics solver often led to very precise trajectories and was very robust at solving our optimization problem. We were slightly concerned over the solve time and its ability to create realistic solutions, so we also opted to look into other trajectory generation methods, specifically kinematic trajectory optimization. This would also allow us to have more control over parameters such as trajectory duration. Our final optimization problem was as follows:

$$\begin{aligned} \min_{\alpha, T} \quad & T_{traj} + L_{traj} \\ \text{s.t.} \quad & X^{start} = f_{kin}(q_\alpha(0)) \\ & X^{end} = f_{kin}(q_\alpha(T)) \\ & \forall i, X_i^{cut} - \epsilon \leq f_{kin}(q) \leq X_i^{cut} + \epsilon \\ & \forall t, |\dot{q}_\alpha(t)| \leq v_{max} \\ & \dot{q}_\alpha(0) = 0 \\ & \dot{q}_\alpha(T) = 0 \\ & T_{min} \leq T \leq T_{max} \end{aligned} \tag{2}$$

where ϵ represented the bounds of accepted solved pose values, which were for us $[0.3, 0.01, 0.01]^T$ for the position and $\pm\frac{\pi}{26}$ for the orientation. We initially had a lot of difficulty making the solver robust enough to not fail. We tackled this issue with several techniques:

1) Our ϵ values were too tight initially, but since we realized a fruit could be sliced along any part of the blade of the katana, we broadened the x bound in ϵ to allow for a larger range of solutions.
2) We initially had orientation constraints for the start and end pose as well as a "pre-cut" point that was 0.1m before the "cut" point as a means of ensuring the blade was travelling with the correct orientation even before it hit the fruit. This "pre-cut" point was removed as it showed no benefit, and the orientation constraint for the start pose was removed, thus speeding up the solver and also making it more robust.
3) The inverse kinematics equation from above was used to solve for a set of q for the first cut pose, then passed in as an initial guess to make the SNOPT solver more robust.

These edits helped the solver improve its chances of finding a valid trajectory. Specific results will be discussed in the upcoming section.

One important thing to note was that while these optimization methods generated valid joint-space trajectories, neither was inherently time-based in the way we structured the optimization problem. That meant that if the robot were to execute the trajectory at time $t = 0$ when the fruit was launched at time $t = 0$, there was no guarantee that the fruit would be intercepted. Thus, we had to add a second step after the optimization problem was solved where we walked through all the points of the trajectory and found the closest point to where the fruit was predicted to land. We then extracted the time t_c at which the sword would pass through that point, then compared that to the time the fruit was predicted to land there t_f. Given the array of times T from the solver, we then shifted all times as such:

$$\forall i, T[i] = T[i] + (t_f - t_c) + dt$$

Essentially, we added buffer time to the trajectory such that the robot would wait to execute the trajectory so it

wouldn't miss the fruit. Note the term dt was added so that the robot would be beside the fruit at the predicted intercept point and not under it so that an actual slice could occur. Through experimentation, we found that $dt = -0.05$ was a good value that gave good success for both solvers.

IV. Results & Discussion

IV.1. Perception System Error

To evaluate the performance of the perception algorithm the error of the estimated centroids was found. The error function was defined as follows: $E = \frac{1}{3}\sum_{i=1}^{3}|C_{a_i} - C_{m_i}|$ where C_{a_i} represents i'th element in the actual centroid position vector and C_{m_i} represents the i'th element in the measured (algorithm output) centroid position vector. By finding the error for 20 random configuration of fruits, the average error was found to be 0.035m. This error is approximately 4 times smaller than the 0.15m diameter used for the fruits within the program. Therefore, it is likely that the deviation between the perceived and actual positions of the fruit will not have any noticeable effect on the ability of the robot to slice the fruit. The standard deviation for these results was 0.000611 and so the error is very consistent and insensitive to the position of the fruit relative to the camera.

It was also found that increasing the size of the fruit did result in an increase of the error. However, the increase in error always remained small in comparison to the size of the fruit. For example, with a 0.4m diameter fruit the error doubled to 0.07m, but this is still over 4 times smaller than the diameter of the fruit, and so it can be extended that for various positions within the environment and for a large range of fruit sizes, the algorithm's error is small enough to allow accurate simulation.

IV.2. Perception System Speed

Another metric used to evaluate the perception algorithm was its speed. To test the speed of the algorithm, the execution time was measured for an increasing number of fruits within the images. 5 trials were taken for each number of fruits and each trial was averaged. The results are shown below:

From the graph it is evident that there is a linear relationship between the number of identifiable fruits and execution time. It can also be seen that for 50 fruits, the approximate execution time is 0.3 seconds. Since a person and the robot would struggle with even 5 fruits being launched simultaneously, it becomes clear that the algorithm is efficient enough to process images at the desired sampling rate and will not bottleneck the simulation.

IV.3. Solver Trajectory Dynamics

We ran a set of 20 trials—throwing a fruit at a random location 20 times for the two solvers—to evaluate their performance (after 20 trials our notebook ran out of RAM due to all the data we were storing). We graphed the output velocities of the katana using the Jacobian of the robot. From making these plots, the overall shape of the velocity curve made sense to us. For a brief period of time during the simulation period, the end effector velocity would spike, indicating it was executing the trajectory. We observed some noise in some of the graphs, and when watching the simulation, noticed that it was likely due to a messy collision with the split fruit halves, as in the collision with the fruit would sometimes cause the katana to move around rather than have a clean cut.

Another interesting observation was that on average, the inverse kinematics solver had a much higher velocity than the kinematic trajectory solver. We suspect this was due to the fact that there were no velocity or time constraints when we set up the optimization problem. This was one of our concerns with using inverse kinematics, specifically that it would produce outrageous solutions such as moving the katana at over 25 m/s.

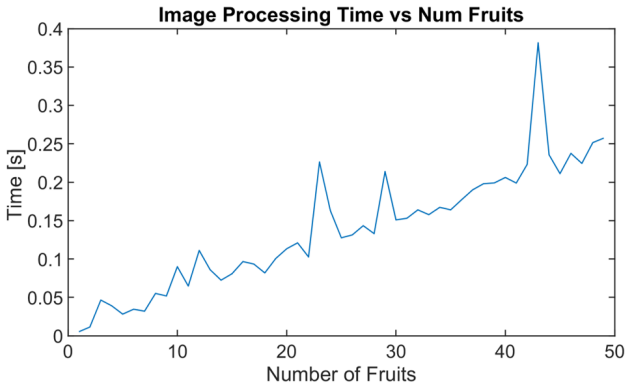

Figure 8. Graph displaying the execution time for the perception algorithm for a given number of fruits to identify and process

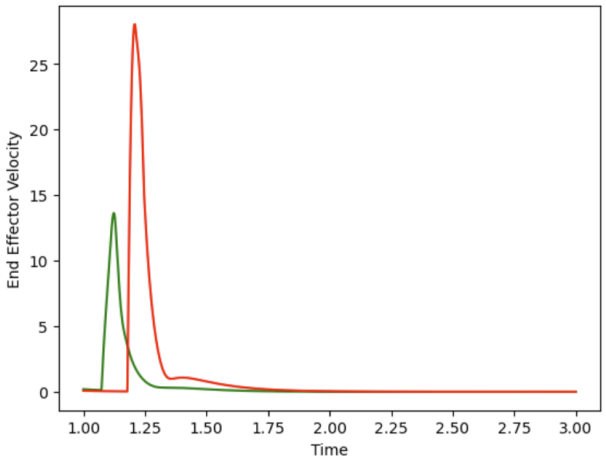

Figure 9. Katana velocities for inverse kinematics solver in red and kinematic trajectory optimization solver in green.

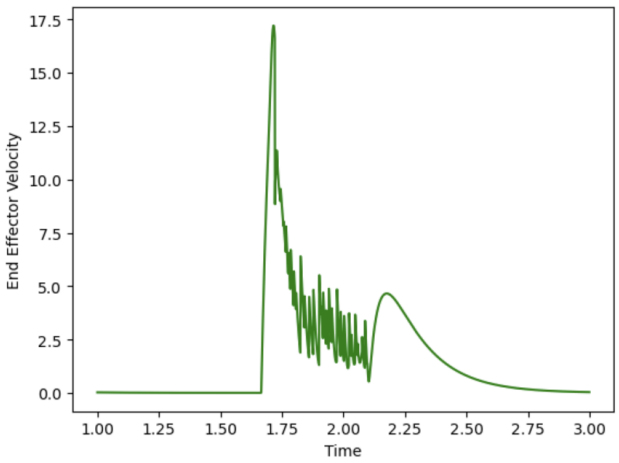

Figure 10. Katana velocity for kinematic trajectory optimization solver (inverse kinematics solver did not succeed in this trial. Note the noise that was caused by the katana colliding with the sliced fruit more than once.

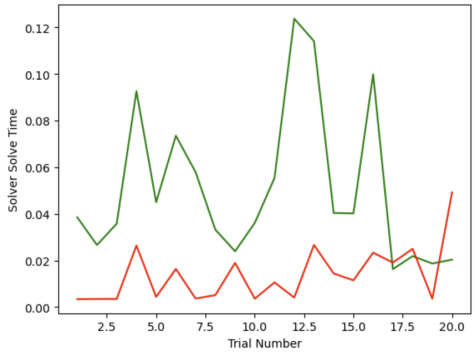

Figure 11. Solve times per trial for inverse kinematics (red) and kinematic trajectory optimization (green).

IV.4. Solver Performance

Solver	Overall Accuracy	Solver Success Rate	Fruit Hit Rate
IK	45%	45%	45%
KTO	75%	100%	75%

Comparing the two solvers in our trial of 20 fruits, we noted that inverse kinematics performed much worse in solving for a trajectory. However, kinematic trajectory optimization was very good at solving for a valid trajectory, but not as effective at generating a trajectory and executing it to hit the fruit. This could also be a function of our timing step where we shift the trajectory in time, and would thus require tuning dt or solving for dt in a more analytical way.

Overall, the solve times for both solvers were well within the bounds for real-time cutting of fruits. We noted that inverse kinematics was on average much faster, perhaps due to the lesser number of constraints and the fact that kinematic trajectory optimization used inverse kinematics to generate an initial guess.

V. Conclusion

Overall, our system was able to demonstrate cutting a fruit in real-time with some promising preliminary results. However, there are many limitations to the system. We showed that kinematic trajectory optimization appeared to generate successful cutting trajectories most of the time and enabled the robot to cut fruits in mid-air. We showed that inverse kinematics was also successful in generating valid trajectories, but was hindered by the success rate of its solver. Tweaking both the constraints and cost functions of these solvers can certainly increase their reliability. We also created a good perception system that was able to detect different types of fruits and predict their trajectories. This was instrumental for adding realism to our system.

On the topic of realism, future work we would want to implement would be to be more rigorous in how we frame the physics of cutting. Due to the lack of literature and time, our model for what forces and speeds are required to cut a fruit were a bit nebulous. Future work would involve being more concrete on these dynamics requirements and integrating those requirements into our solvers to create more realistic solutions.

In addition, one final point of future work would be to increase the number of fruits thrown at the robot to add more complexity to the optimization problem and cutting trajectory algorithm. This would make our system resemble the fruit ninja game more closely.

VI. Contributions

Billal worked on the perception system. This involves designing and coding the algorithm and implementing it within the project. Ryan worked on adding fruits to the simulation and generating ballistic trajectories for them. He also handled all collision detection. Michael worked on setting up the KUKA iiwa robot and the katana in the simulation. He also wrote the cutting trajectory generation algorithm as well as both inverse kinematic and kinematic trajectory optimization solvers.

References

[1] S. Kim, A. Shukla and A. Billard, "Catching Objects in Flight," in IEEE Transactions on Robotics, vol. 30, no. 5, pp. 1049-1065, Oct. 2014, doi: 10.1109/TRO.2014.2316022.

[2] X. Mu, Y. Xue and Y. -B. Jia, "Robotic Cutting: Mechanics and Control of Knife Motion," 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 2019, pp. 3066-3072, doi: 10.1109/ICRA.2019.8793880.