HARVEST

An autonomous fruit-picking 6-axis robotic arm.

Problem

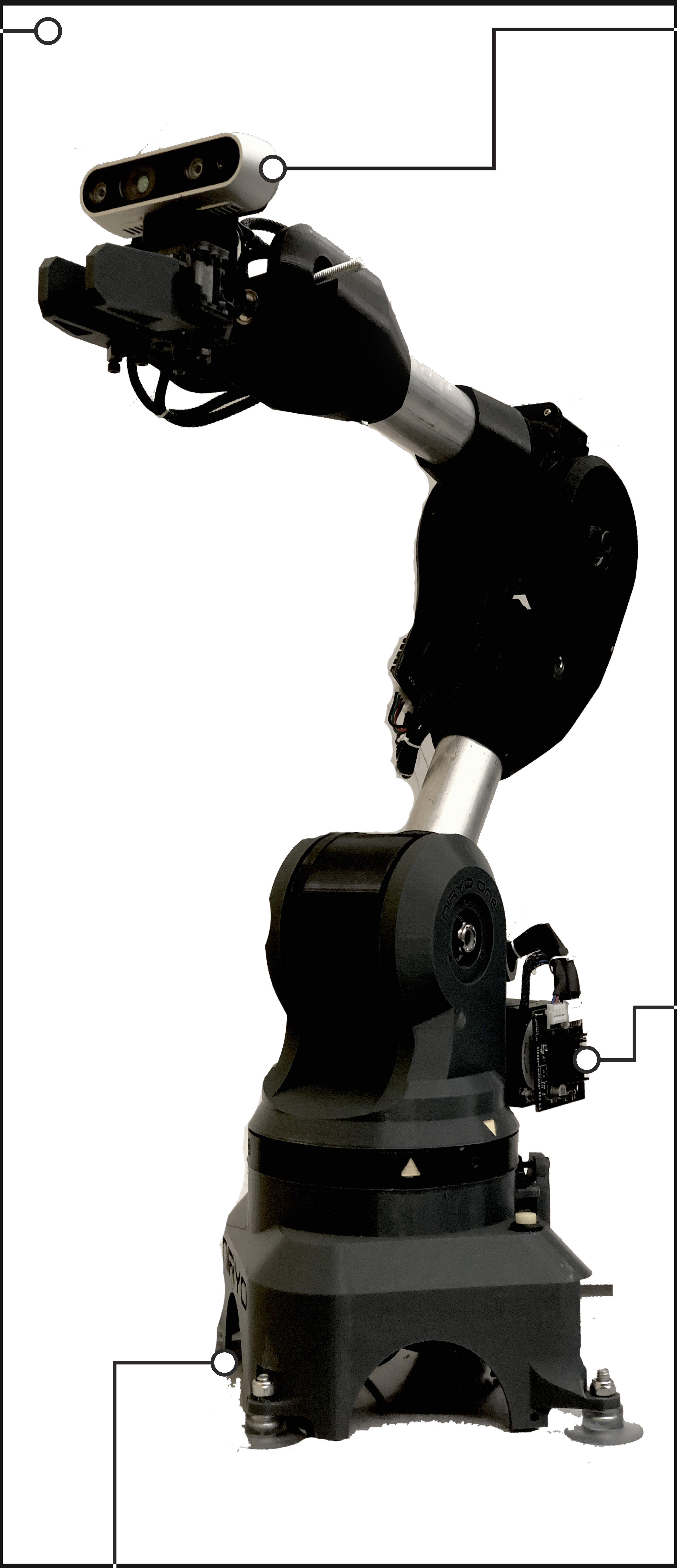
The agricultural industry is facing labor shortages that are aggravating every year. With the continuous increase in the global food demand, we will be faced with severe food shortages in the near future.

Solution

We've utilized a commonly used, versatile 6-axis robot arm with a camera attached and use machine learning to help detect fruits. With a focus on using more ordinary components, we aim to make these tools more accessible for farmers.

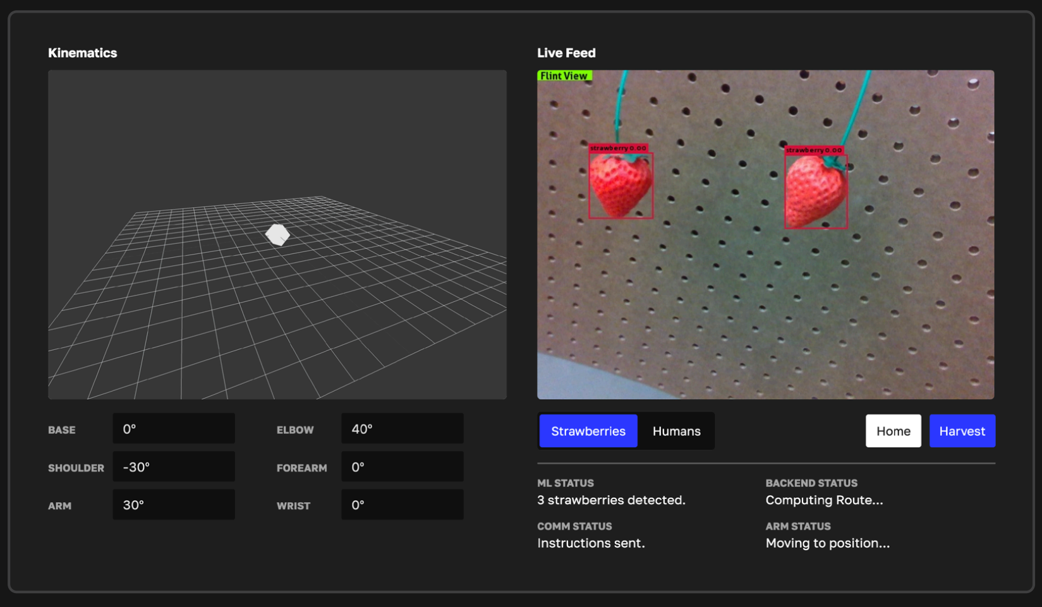
Robotic Arm

The 6-axis robotic arm was built by using the guidelines provided by the open source Niryo One Robotic Arm. The guideline provided the necessary 3D-models for us to fabricate the robot frame using PLA filament on a 3D Printer from scratch and buy all necessary components including stepper motors, servos, gears, etc. The mass of the robotic arm sums up to about 3.3 kg with a maximum reach of 44 cm.



Web Interface

The interface of this robotic arm is a web platform created from Node.js. The website is hosted locally with the frontend written in conventional HTML, CSS, and Javascript and backend using the Express.js framework.



BASE 0°
SHOULDER -30°
ARM 30°
ELBOW 40°
FOREARM 0°
WRIST 0°

ML STATUS
3 strawberries detected.

BACKEND STATUS
Computing Route...

COMM STATUS
Instructions sent.

ARM STATUS
Moving to position...

The frontend sends a `GET` HTTP request to the local server for the arm's configuration at small intervals. When an input is pressed, a `POST` HTTP request is sent to the server which then saves the given instructions to a JSON file for the Python interpreter to parse.

Communication

To communicate between the Raspberry Pi and actuators within the arm, we have implemented the Dynamixel Protocol 2.0 which is the main protocol used by the servos used in the arm.

By using the same communication protocol throughout the entire system, we can daisy chain every component onto a single line, allowing us to communicate rapidly with each of the actuator. This is facilitated through a half-duplex serial TTL communication protocol. The Python programming language is used to enable to serial port on the Raspberry Pi.

The main instructions used in this protocol are the Ping, Read, and Write instructions:

Ping — An instruction to check the existence of a device and basic information.

Read — An instruction to read a value from the actuator.

Write — An instruction to write a value to the microcontroller.

Computer Vision

The computer vision system utilizes two main methods of detecting objects. For one, we use a machine learning-based object classifier called *OpenVino*. This allows us to detect complex objects such as people. To ensure that the model is running at an acceptable framerate, we offload the processing to a Movidius Compute Stick, which accelerates the neural network processing.

For the 2nd method, using some image signal processing techniques, we detected strawberries using a series of filters. The first step is to take the image provided by the Intel RealSense camera and apply a `Hue-Saturation-Value` (HSV) filter to it such that pixel in the image which match the color of a strawberry are highlighted.

With the resulting binary image mask, we used a `Sobel` operator to conduct edge detection and outline the contours of the strawberries. We then filtered these contours for the appropriate size and aspect ratio for a strawberry.

This style of object detection require hand tuning, but is faster and more reliable than any existing neural network-based models. This method is much more reliable and faster than using a neural network in practice.

Stepper Controllers

To ensure a fluent and cohesive communication between the Raspverry Pi and the robotic arm, we have custom built controller boards on each of the stepper motor. Each board containing a `STM32F103C8T6` Microcontroller (Blue Pill) to interpret the incoming instructions, a `DRV8825` High Current Driver to drive the stepper motors, a `AS5047D` High Speed Position Sensor (also called a Magnetic Encoder) for precision sensing, and a `THVD1410DR` Transceiver to moderate the half-duplex communication. The boards are programmed to receive the instruction packets based on the Dynamixel protocol and designed to be addressable so that we can simply daisy chain all of the different actuators into a single cluster of data and power lines.

Interpreter

The interpreter, written in Python, is the central control of the system. It manages the backend connections with the front-end, stores the latest configuration of the in space, and coordinates communication to the individual actuators based on the Dynamixel protocol. The camera feed is processed by the interpreter through the `pyrealsense` library which allows the program to rapidly communicate with the camera to determine relative depth at a specific point in the view. The interpreter takes data from various sources and commands the robot appropriately to achieve its goal of reaching its assigned target.

Kinematics

To convert a target position to 6 angle configurations for the robotic arm to execute, we intergrated an inverse kinematics library `ikpy` within the interpreter. This provides us the necessary computation to determine the angles at which each of the joint should be in order to reach a specific location in space as a target.

ELECTRICAL & COMPUTER ENGINEERING
UNIVERSITY *of* WASHINGTON

Doruk Arisoy
Nick Huber
J. Brandon Iams
Khang Lee