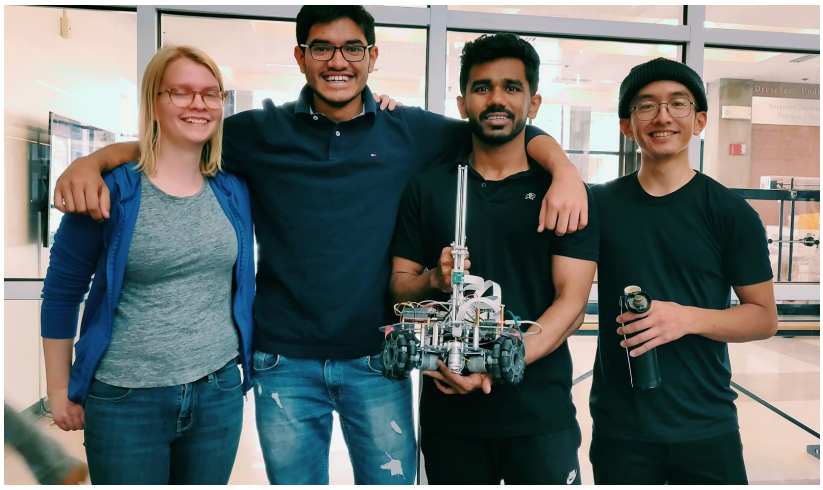
Automated Warehouse Robot

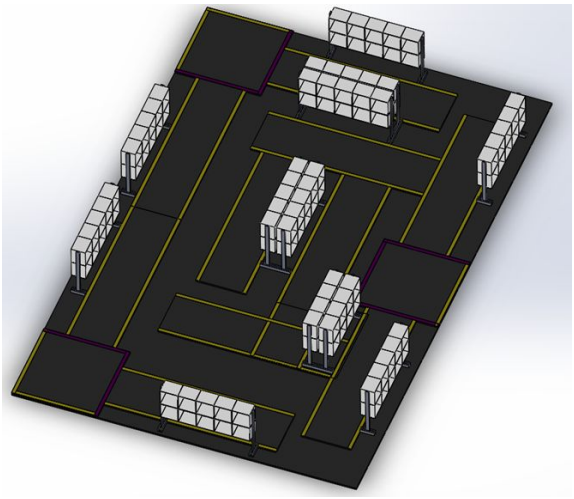


Team Æ s t h e t i c

Arth Beladiya
Aisling Pigott
Karl Sanchez
Kunal Shinde

Introduction and Mission

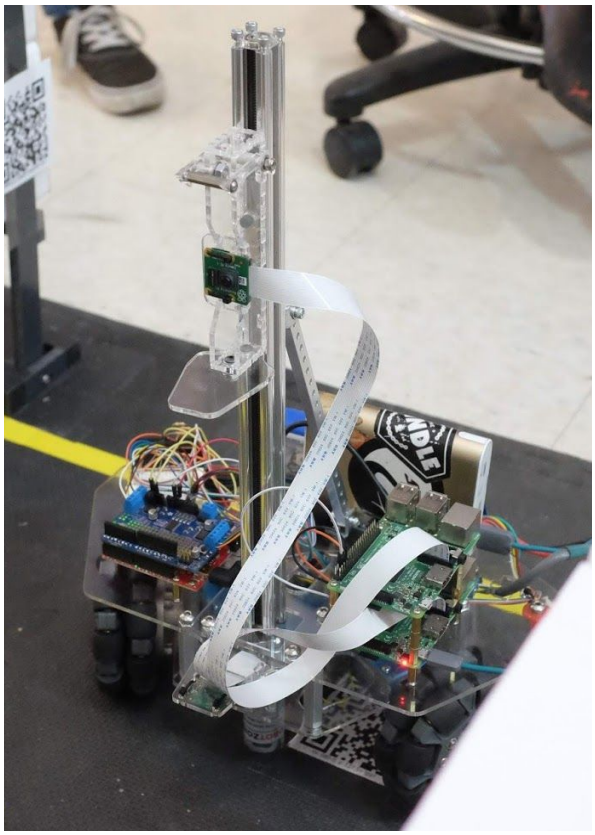
The Automated Warehouse Robot was made as part of a simulated Amazon Warehouse Challenge in the Spring 2019 Mechatronics course at CU Boulder. The goal of this challenge was to design, build, and operate an autonomous vehicle for retrieving and delivering packages in a mock-warehouse playing field. The vehicle starts in a designated area, a "loading bay," where it is shown a QR code describing a target pallet and a destination loading bay. The robot must then, fully autonomously, navigate to the requested pallet in the warehouse, extract this pallet from a rack, and deliver it to a loading bay by dropping it off on a small platform.



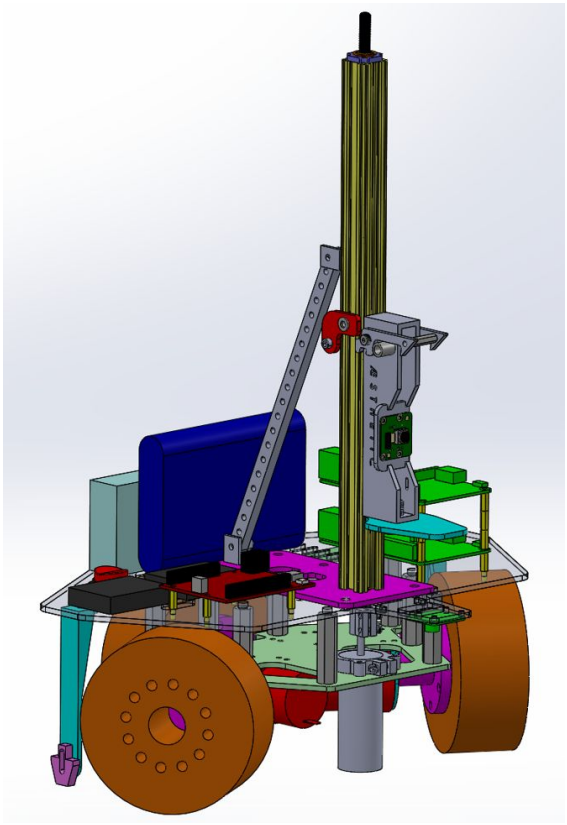
Automated Warehouse Challenge Playing Field

Robot Overview

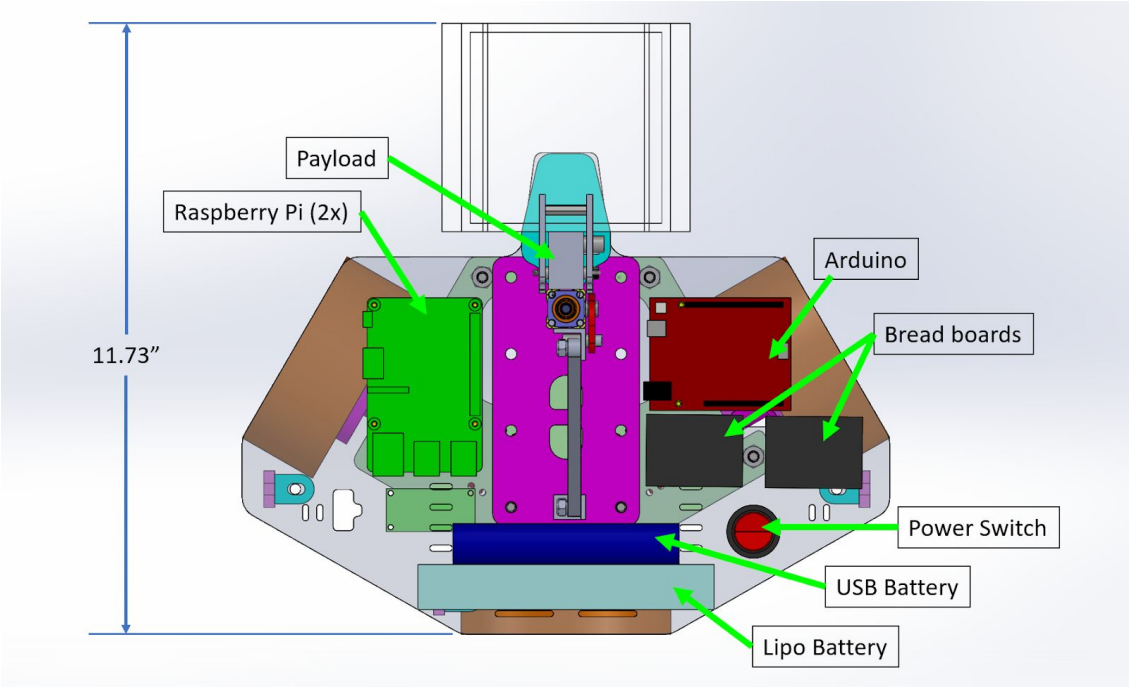
Here we give a brief overview of each subsystem and how it relates to the integrated whole. The robot drivetrain encompasses the motors, wheels, and associated power delivery system. This robot uses three Jameco DC-gearmotors, each driving an omni wheel. The payload subsystem retrieves and places the pallets from the rack and loading bay, respectively. To handle multiple levels of pallet retrieval, the system uses a lead-screw driven by a dc motor. Attached to the lead screw nut is the pallet lift assembly, a device which hooks onto the pallet on pickup and releases on drop off. All four motors are powered by a lithium polymer battery and controlled by an Arduino Uno microcontroller and accompanying AdaFruit motor shield. An array of sensors close the feedback loop during navigation and pallet handling. The robot has two QRB1113 infrared reflective sensors for line sensing and two Pi-Cameras for reading ground and wall mounted QR codes. All command and data handling is done by a single Raspberry Pi (the GroundPi) while a secondary Pi (the SkyPi) exclusively handles the input from the second Pi-camera. Both Pi's run python scripts which handle the mode execution.



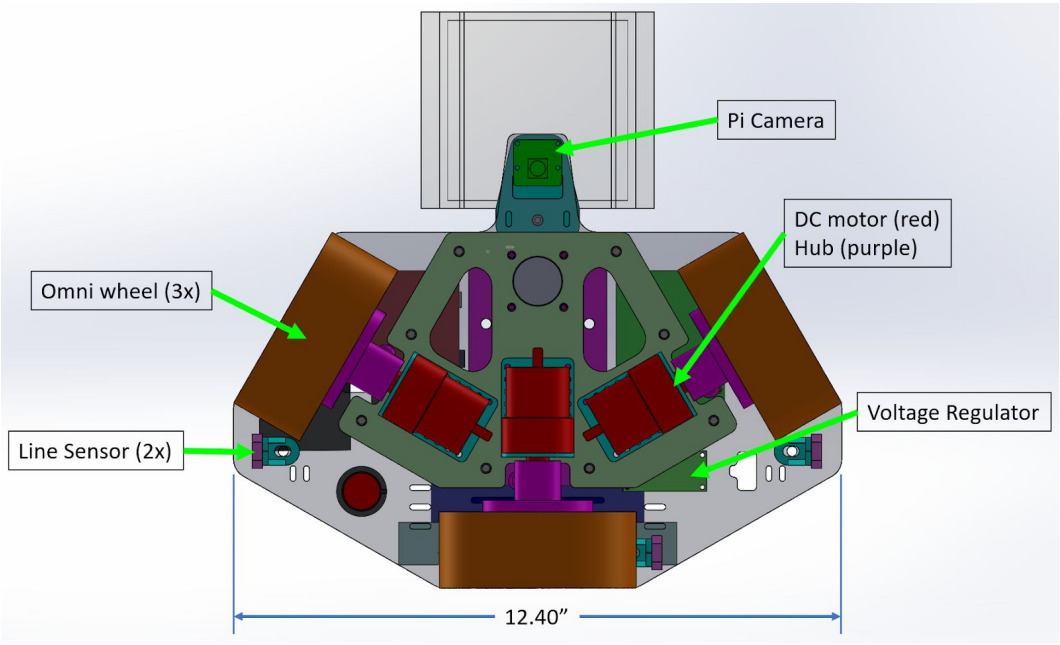
Completed Robot



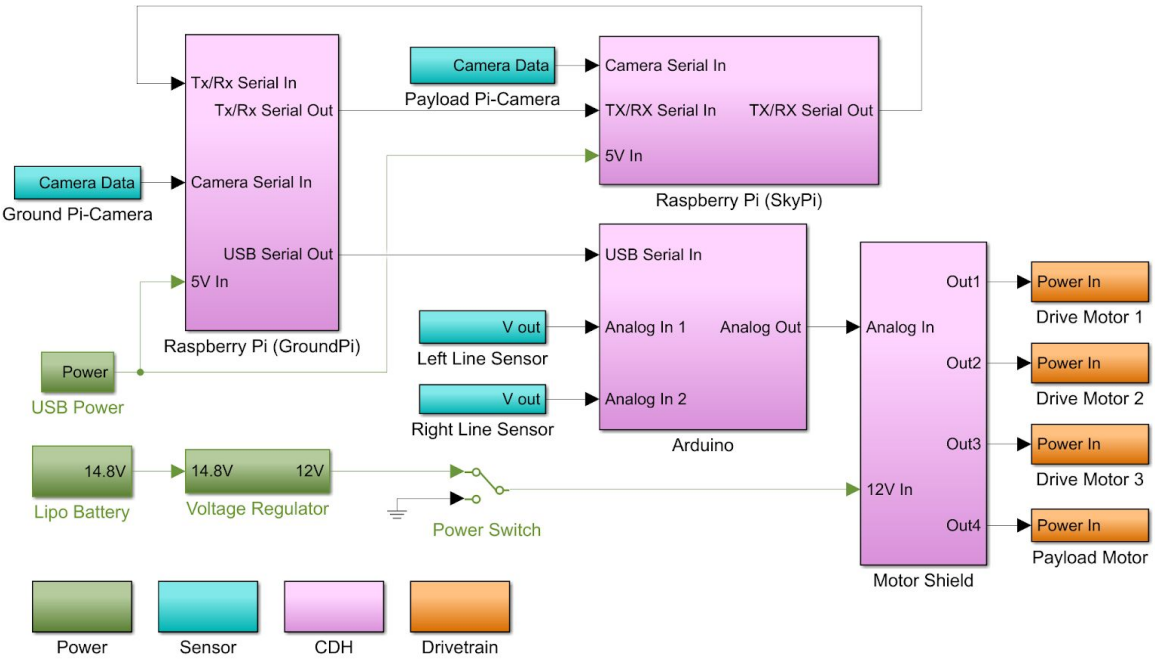
CAD Assembly



Top View Component Diagram



Bottom View Component Diagram



Hardware Block Diagram

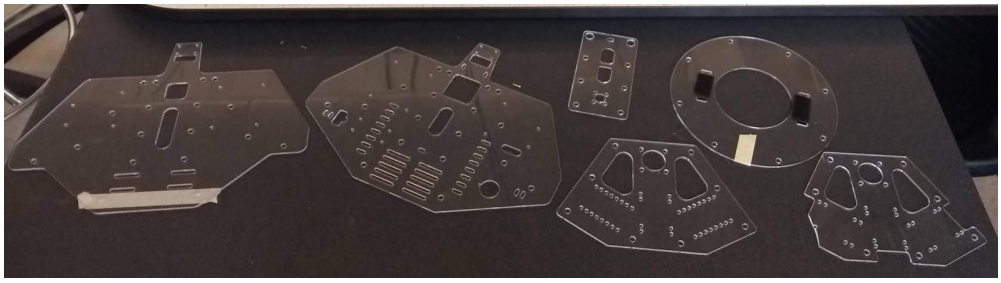
Design of Subsystems

Chassis Design

The chassis subassembly encompasses all components not included in the payload subassembly. Fundamentally, the chassis is made of a sandwich of 3 mm acrylic plates separated by 80 mm standoffs. The team chose this two-tier design to give the robot stiffness and ample mounting surfaces for components. The lower tier serves as the mounting point for

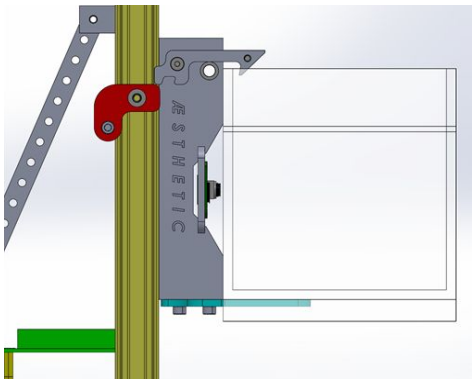
the drivetrain subassembly, which consists of three omni wheels driven by DC motors. The upper tier is the mounting surface for the rest of the components: the payload, batteries, and all circuit boards.

Laser-cut acrylic was chosen for its durability and ease of manufacture. The holonomic drivetrain design requires accurate angular alignment between the motors and the sheer number of components on the vehicle require many mounting holes. Both of these are quickly achieved with laser cutting. As can be seen below, laser cutting also allowed for rapid prototyping of several designs.

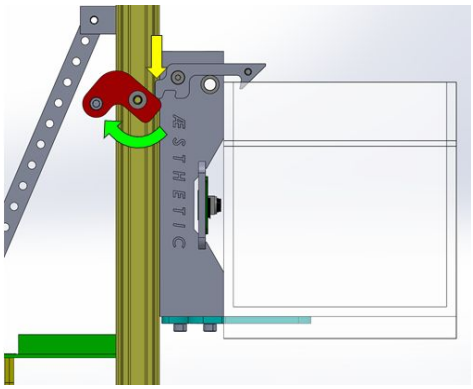


Revisions to chassis design

Payload Design



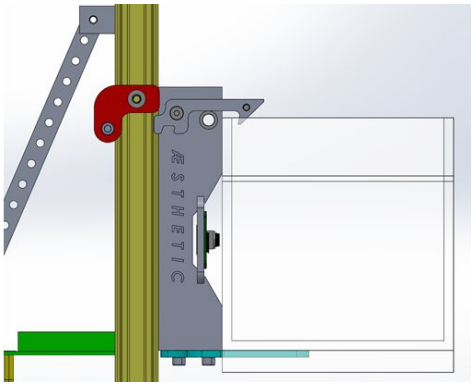
High Position



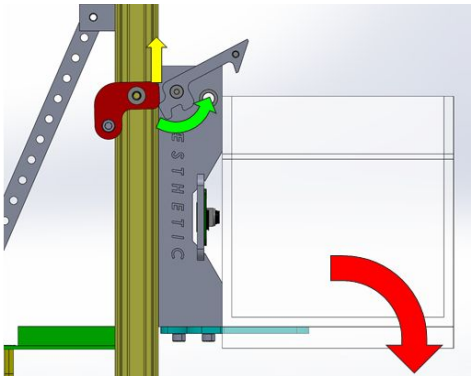
Latch allows free downward movement

The payload subsystem is responsible for pallet extraction and delivery. To retrieve pallets on different rows, the team used an off-the-shelf lead screw system for vertical positioning. The lead screw is driven by a DC motor on the bottom of the robot and is mounted inside a hollow length of railed aluminum extrusion with flanged ball bearings on both ends to constrain its location. A specially designed lead screw nut slides axially along the length of the extrusion as the screw is driven. Tapped holes in this nut provide the mounting surface for the pallet lift.

The pallet lift, as the interface between the robot and its cargo, is subject to several design constraints. Foremost is the clearance between the shelf and the pallet itself. The lift must grip a pallet which has less than 0.4 inches of clearance on any given side. This low clearance translates to an accuracy requirement for lift alignment in both the horizontal and vertical axes. The final design relaxes these alignment requires as much as possible by virtue of a passive mechanical hook and lower lift plate.



Low Position



Latch releases payload on upstroke

The hook design takes advantage of the open pallet top by hooking over the pallet's front face while a plate at the bottom supports the pallet. The plate was sized such that the pallet's center of gravity is unsupported. Thus, the natural pitch-forward moment of the pallet keeps it in contact with the hook. Pallet release is handled by a one way latch (in red). As seen on the left, this latch allows the pallet lift to travel down without releasing the hooks. However, once the lift moves below a critical point, the latch swings down and locks. As the pallet lift travels upward toward the now-locked latch, the hooks will swing up, releasing the pallet onto its loading dock. Thus, this design allows the robot to pick up both high and low pallets and release them without any extra motors or servos other than the vertical lift motor.

The hook is sized such that it can engage the pallet over the 0.375" vertical range of possible locations of the lower lift plate (in blue) thereby decreasing required vertical accuracy. Horizontal alignment error is addressed by the assembly's narrow width; the hooks and support plate both engage with more than 1" of misalignment.

Previous iterations of Payload Design

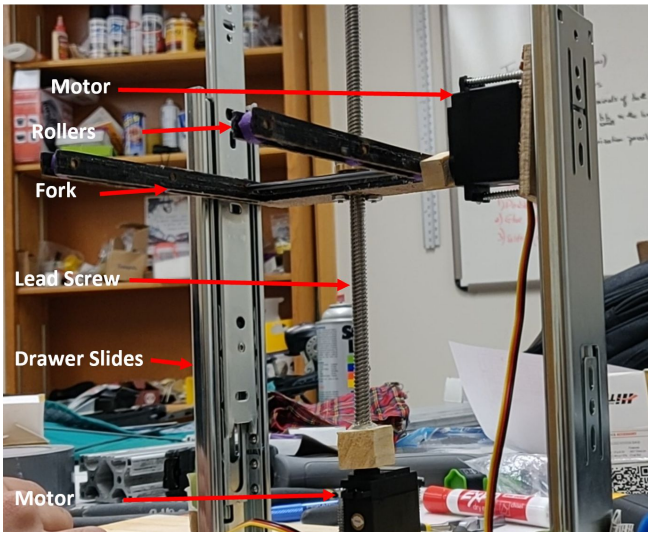
Iteration 1

For the first iteration, the team decide to proceed with a forklift mechanism. The fork was made of laser-cut wood which used a lead-screw mechanism for vertical motion. Drawer slides were used to support and align the fork. There were six rollers which were supposed to be attached to a belt-drive system which would draw the pallet towards the robot once the fork slides under the pallet. There were several drawbacks to this design. First of all, the drawer slides were too heavy and did not support vertical movement. We realized it later that they were supposed to be used only in horizontal movement. In addition, the missing coupler made the vertical movement of the fork imprecise. The reduced tolerances in the racks also made it difficult to place the rollers in the fork.

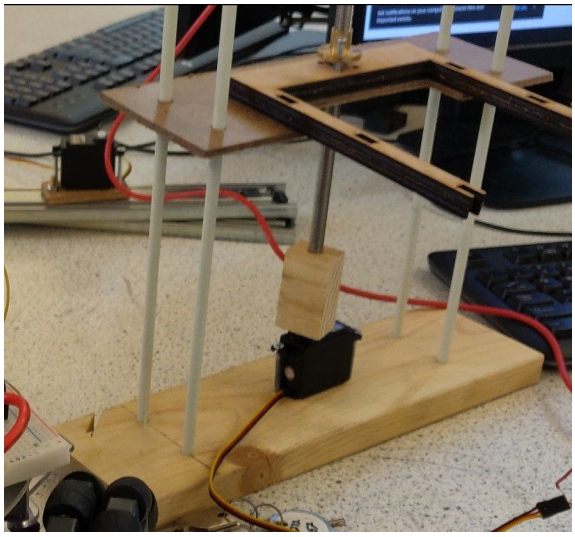
Iteration 2

The drawer slides were replaced by four stiff plastic columns which made the system lighter. The missing coupler between the motor and the lead screw still caused inaccuracies in the vertical movement of the fork. The original idea of the belt drive to pull the pallet in was

scrapped. In the renewed design, the fork would slide under the pallet, pick it up and move back using the main drivetrain.



Iteration 1



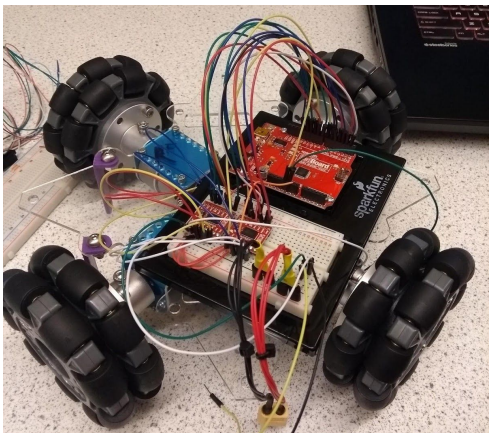
Iteration 2

Drivetrain Design

The team's choice of a holonomic drivetrain was driven by the mission constraints on maneuverability. Since the pallets must be removed from shelves not much larger than the pallets themselves, the robot must be able to make small, precise lateral maneuvers to align its payload to the pallet. Additionally, rotational movement was necessary to orient the robot towards the pallet rack while in the aisle. Thus, it made sense to have a drivetrain system that would decouple translational from rotational motion. Specifically, we wanted the robot to strafe left and right without having to drive forward and turn, as a skid-steer drivetrain would necessitate.

The final drivetrain uses 100 mm omni wheels paired to aluminum interfacing hubs which bolt directly to the motor shafts. The motors themselves are mounted to the acrylic with Pololu motor brackets. Each of the three wheel rotation axes are aligned 120 degrees apart, as is the norm with holonomic drive trains. However, the rear motor had to be mounted closer to the intersection between all three axes due to aisle width constraints. As a result, traversing (left or right translation) and rotational maneuvers involved increasing the commanded PWM speed for the rear motor to compensate for its shorter distance to the center.

The team originally chose a 4-wheeled design to maximize the robot's stability while simplifying control design. However, we later switched to a 3-wheel design due to changes in the payload design, as discussed in previously. The three wheel design, while less stable and more difficult to control, still offers the benefits of a holonomic drivetrain.



Initial four wheel drivetrain

Power System

The drivetrain is powered by a 4000 mAh 4S 30c lithium polymer battery, which outputs 14.8V at full charge. Since the motors and motor shield can only run up to 12 V, the battery voltage is controlled by a step up / step down voltage regulator. As the battery voltage drops during discharge, the regulator smooths this output to a constant 12 V. A two-position rocker switch turns the power from this LiPo battery to all motors on and off. All computers and microcontrollers are powered by a USB battery bank that outputs a constant 5 V. The team's choice to decouple the power supplies for the computers and the motors allows for testing of the software logic without actually having the motors run and provides more consistent voltage to the Raspberry Pi's which are more sensitive to voltage irregularities.

Sensors and Control Handling

To aid in autonomous navigation through the warehouse, the robot is equipped with two line sensors and two Raspberry Pi cameras. The line sensors allow robot to maintain its wall clearance as it traverses the aisles while the pi cameras read the QR codes scattered throughout the environment. One Pi camera faces the floor to read coordinate QR codes while the second Pi camera, mounted to the pallet lift, reads pallet QR codes and aids with alignment during pallet extraction.

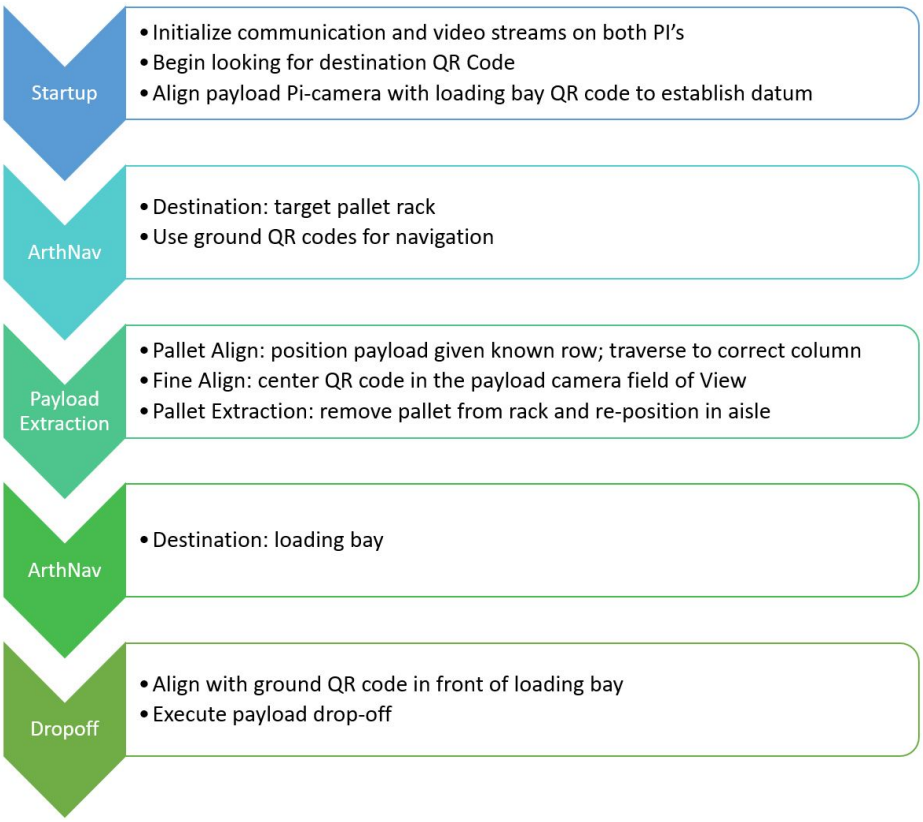
The robot uses two Raspberry Pi's and one Arduino for logic and motor control. The two Pi architecture allows for processing two video streams (using two Raspberry Pi-Cameras) which was necessary to handle both the course and pallet alignment.

The first Pi which navigates the course was given direct USB serial communication to the Arduino which the motors through an Adafruit I2C motor shield. The second Pi which which was originally intended to handle pallet alignment passes information to the Arduino through the first Pi via Tx/Rx serial communication. The Arduino was left to handle feedback from the two infrared reflectance sensors which keep the robot within the aisle and communicate that information to the motors directly.

Software Architecture

The robot's software architecture follows from its mission profile. Each distinct segment of the mission represents a mode of operation for the vehicle, with specific conditions transitioning the robot from one mode to another. The diagram below describes these modes of operation which, for this relatively simple mission, flow approximately sequentially as the robot goes about its tasks.

The modes, represented by segments on the diagram, are implemented as Python functions which are run on either or both of the Raspberry Pi's depending on the mode. This is necessary because the sensor data from the payload camera, which is attached to the SkyPi must close the loop with GroundPi, which is connected to the motors. A more detailed description of these modes' implementation can be found in the Appendix.



Software Mode Diagram

Operational Modes

Path Planning

The path planning algorithm as seen in the Appendix listing for "ArthNav.py" works as follows: each intersection is modeled as a dictionary with keys 'xy' and 'rel'. The 'rel' (stands for relative) key defines which intersections are directly connected to the host intersection. The first part of the algorithm consists of the path decider (function named 'pathplan'). This

function takes in the start and destination intersection and returns a path to the destination. This is done by calculating the distance of each intersection in the 'rel' key of the current intersection and appending the intersection with the shortest distance to the path. This process repeats until the destination is reached. The second part of the algorithm converts the path into Arduino usable instructions (the 'pathtodirections' function). Using the current orientation of the robot and the required orientation, it determines which turns to make. As the grid is cartesian and all intersections are at right angles, this process is straightforward. The third and final part of the algorithm commands the robot to move in real time. This part utilizes an infinite loop, the path list and the directions list. To determine if an intersection has been reached, the robot continuously scans to the ground QR codes and updates its current location in the algorithm when a QR code is scanned.

Payload Loop: Alignment

It was initially our intention to use the SkyPi and camera to align the robot with the target payload. Using Pyzbar the Raspberry Pi creates a bounding rectangle for all scanned QR codes. The centroid of the QR code can be determined with these bounding coordinates and the robot can be aligned accordingly. The scale of the bounding rectangle is then used to determine approximate distance from the target pallet and move the robot towards the pallet before starting a time based extraction. Due to time constraints the final software does not allow for communication between the two Raspberry Pi's although a beta version of this process can be seen in the Appendix.

The final software uses the pallet rack QR code at the left end of the pallet rack as a starting point for a time based alignment process. Using the pallet column information given at the start of the course the robot traverses x seconds for each pallet column index before starting the pallet extraction process.

Payload Loop: Pallet Extraction

Once aligned with the target pallet the robot is instructed to move forward into the pallet. Due to the field of view of the pallet camera this process must be conducted blindly relying simply on a timer. This process is aided by the passive pallet arm mechanism that maximizes the misalignment tolerance. The robot is then instructed to back up before re-entering the navigational loop for drop off.

Payload Loop: Aisle Re-alignment

After the target pallet was lifted onto the pallet arm the SkyPi connection was closed. The robot was then directed by the GroundPi to the drop off loading bay using the initial path as determined in ArthNav.

Drop Off

Once the robot reaches the designated dock, it goes through a series of time based maneuvers based specific to the layout of the destination dock, aligning itself in front of the drop-off point. The initial time based alignment allows places the robot within the field of view

of the drop off QR code which the GroundPi uses to align the robot with the drop off point. This alignment process is nearly identical to the QR alignment of the pallet QR code. Following this it drops the pallet and backs up.

Lessons Learned

1. *Budget about 30% of the course to integration and testing.*

Throughout the course the team experienced several delays due to longer than anticipated integration and testing phases. We were able to modify our first major design iteration (the forked arm pallet lift) to pass the initial systems integration phase by implementing a timed pallet pick up routine. Although this first test was successful in passing the test we realized later that we had to manually position the robot within tighter tolerances than we could achieve autonomously. The resulting redesign phase took longer than anticipated and further reduced the amount of time we had to integrate and test the final systems.

2. *Design code in a modular way to make testing of individual parts easier.*

At the end of the systems integration phase we realized that the code was not designed to be run in segments. Since we were unable to implement a framework that integrated individual functions of the robot we ended up using the navigational code as a framework. Although the final code was mostly functional it could have been improved with a framework that allowed for individual testing of navigation, pallet alignment and pick up, and pallet drop off. This modularization would have likely aided in Pi to Pi communication which proved to be a challenge in the final software.

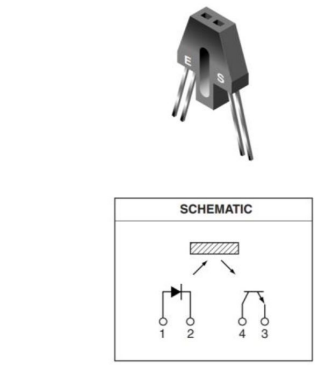
3. *Push for tight tolerances on the playing field.*

Other difficulties were enhanced by imperfections in course construction. One of the major issues with the pallet arm mechanism was that it was designed to meet the course specifications for pallet rack size. However, once we were able to test the final pallet arm mechanism we realized that the pallet rack slots were slightly smaller than spec. Due to this discrepancy our pallet arm would bind on about 50% of the pallet slots. Other issues of course construction were overcome by design: the use of DC rather than stepper motors meant that our robot was easily able to accomodate large tolerances of course layout and varying pallet rack heights.

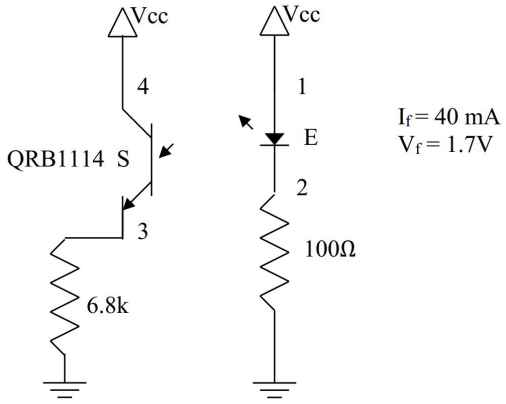
Appendix

Circuit Diagrams

The line detectors used circuits from a previous lab, as shown below. The output from the transistor (wire 3) was connected to the Arduino via analog inputs. Other circuits are integrated into the off-the-shelf components we used and can be referenced via their specific product pages.



1 = Red, 2 = Green, 3 = Blue, 4 = White



Code Listings

Mode information is exchanged using serial commands between the two Raspberry Pi's encoded in Ascii as number strings. The Arduino code runs the motors according using a similarly encoded set of instructions between itself and the the GroundPi. Both sets are shown below.

Serial Command	Description	Context
0	Rotate right	GroundPi to Arduino
1	Rotate left	
2	Traverse right	
3	Traverse left	
4	Move forward	
5	Move reverse	
6	Payload up	
7	Payload down	

8	Turn off all motors	GroundPi to Arduino
9	Line follow and go forward	
20	Read QR	GroundPi to SkyPi
21	Execute Payload Loop	GroundPi to SkyPi
22	Execute Drop-off Loop	GroundPi to SkyPi
28	Success signal for SkyPi functions	SkyPi to GroundPi
29	Mission end signal	GroundPi to SkyPi

GroundPi Main Code

```python
# MCEN 5115 - Automated Warehouse Robot
# Authors:
#   Aisling Pigott
#   Kunal Shinde
#   Arth Beladiya
#   Karl Sanchez

# Description:
# STATEMACHINE is the main python script for our robot.
# All modes of operation are encapsulated in functions
# that return to this code upon completion. Note that all
# functions in this code are run on the bottom (Master) Pi.
# ------------------------------------------------------------

# Import mode files with respective functions
import startup
import ArthNav
import payload
import dropoff
import serial
import pi_init

# Call mode function and handle mode switching via serial
def runState(mode,pi_ID,arduino_ID):
    pi_ID.write(mode.encode("ascii"))
    success = False
    while not success:
        cmd = int(pi_ID.read())
        # Listen for success
        if cmd == '28':
            success = True
        # Stream commands to Arduino
        else:
            arduino_ID.write(cmd.encode("ascii"))

    # Tell Arduino to stop activity
    arduino_ID.write('8'.encode("ascii"))
    # Tell SkyPi to stop activity
    pi_ID.write('0'.encode("ascii"))

# INITIALIZE ------------------------------------------

# Turn on the ground pi serial and camera
skyPi, vs = pi_init.start('/dev/serial0')

# Turn on arduino
arduino = serial.Serial('/dev/ttyUSB0',74880,timeout=.1)
arduino.flushInput()

# BEGIN MISSION ---------------------------------------

# Read target QR code
skyPi.write('20'.encode("ascii"))
success = False
while not success:
    # Listen for success
    if cmd == '28':
        success = True
    # Listen for QR code
    else:
        QR = int(skyPi.read())
        print(QR)

# Setup pallet alignment ------------------------------
runState('24',skyPi,arduino)

# Begin Navigation to Pallet rack ---------------------
ArthNav.main(vs, QR, arduino, 0)

# Execute Pallet extraction ---------------------------
runState('21',skyPi,arduino)

# Continue to Loading bay -----------------------------
ArthNav.main(vs, QR, 1)

# Begin Dropoff Loop ----------------------------------
dropoff.main(vs, arduino)

# END MISSION -----------------------------------------
skyPi.write(29.encode("ascii"))
```

SkyPi Main Code

```python
# MCEN 5115 - Automated Warehouse Robot
# Authors:
#   Aisling Pigott
#   Kunal Shinde
#   Arth Beladiya
#   Karl Sanchez

# Description:
# STATEMACHINE2 is the main python script for the sky Pi.
# The sky pi functions as a slave device to the ground pi,
# which is physically located on the bottom of the pi stack.
#
# We manually start *both* stateMachines at the beginning of a
# run. This code immediately opens up a serial port and starts
# listening for commands from the master pi.
# Commands trigger the functions defined here. All functions will
# run their own internal loops and return a success or a failure
# state.
# ------------------------------------------------------------------

# Import mode files with respective functions
import serial
import qrParse
import pi_init
import payload
import dropoff
import startup

# Start up the sky pi
groundPi, vs = pi_init.start('/dev/serial0')


# Begin listening for commands from ground
# Commands take the form of integer values 20-29

while True:
    cmd = int(groundPi.read())

    if cmd == 20: # Startup sequence: read QR code
        print("Command = ",cmd)
        # target QR is the initial data provided
        target_QR = qrParse.qrParse(vs)
        pallet_ID = target_QR[1]
        dest_ID = target_QR[4]
        groundPi.write(target_QR.encode("ascii"))
        # Give some time for ground Pi to acknowledge
        time.sleep(0.1)
        groundPi.write(28.encode("ascii"))

    elif cmd == 21: # Enter Payload Loop
        print("Command = ",cmd)
        # Find and extract pallet
        payload.main(vs, pallet_ID, groundPi)

        # Give some time for ground Pi to acknowledge
        time.sleep(0.1)

    elif cmd == 23:
        print("Command = ",cmd)
        startup.main(vs, groundPi)
        # Give some time for ground Pi to acknowledge
        time.sleep(0.1)

    elif cmd == 29: # Mission Accomplished; exit code
        break
    else:
        print("sky pi standby")
```

Path Planner Pseudo Code

```
# Description:
# Path Plan is the general warehouse navigation algorithm for our
# robot. Given destination data from a decoded QR code, Path Plan
# will drive the robot there.
# Note that there are to modes to Path Plan:
# Mode 0 = payload destination -- the algorithm will place the
# robot facing the target pallet rack while horizontally located
# on the left-hand side of the rack
# Mode 1 = loading bay destination -- the algorithm will steer the
# robot to the loading bay before transitioning to the dropoff mode

function pathplan(start,destination,intersections):
  path = []
  while(current_intersection not destination):
    # calculates distance of all intersections in 'rel'
    # of current_intersection to destination
    # appends intersection with shortest distance to path[]
  return path

function pathtodirections(start_orient,path):
  directions[]
  # append arduino instructions as characters to directions[]
  # based on manoeuvre to perform
  return directions
```

```
function main(videostream,initial_location,QR_data,mode):
  # Initialize serial connection to Arduino
  if mode==0: #find pallet
    start = initial_location
    destination = racklocation from QR_data
  elif mode==1: #return pallet
    start = racklocation
    destination = dock_from_QR_code

  path = pathplan(start,destination,intersection_list)
  directions = pathtodirections(start_orient,path)

  while True:
    # continuously scan for ground QR codes
    if QR_code_ground == found and nextIntersection not reached:
    # Follow directions[currentIntersection]

    elif QR_code_ground == found and nextIntersection == reached:
    # Follow directions[next intersection]
    currentIntersection  = nextIntersection

    if destination reached:
      break
    # When destination is reached
    # Turn towards rack
```

Arduino Motor Driver Code

```
/* Authors:
 *    Aisling Pigott
 *    Kunal Shinde
 *    Arth Beladiya
 *    Karl Sanchez
 * This code will control the 3 motors of the chassis and the pallet motor
 * as directed by serial input
 */

#include <Adafruit_MotorShield.h> //motor shield is a slave device of the Arduino
#include <utility/Adafruit_MS_PWMServoDriver.h>

Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *backMotor = AFMS.getMotor(1);
Adafruit_DCMotor *rightMotor = AFMS.getMotor(2);
Adafruit_DCMotor *leftMotor = AFMS.getMotor(3);
Adafruit_DCMotor *palletMotor = AFMS.getMotor(4);

//Two sensor pins
const int leftSensor = A1;
const int rightSensor = A2;
char current = -1;
void setup() {
  AFMS.begin(); //controls the Adafruit motor shield

  Serial.begin(74880);

  backMotor->setSpeed(50);
  leftMotor->setSpeed(50);
  rightMotor->setSpeed(50);
  palletMotor->setSpeed(50);

  pinMode(A1,INPUT_PULLUP);
  pinMode(A2,INPUT_PULLUP);
}

void loop() {
  int sLeft = analogRead(A1);
  int sRight = analogRead(A2);
  // analog values greater than this threshold mean that tape is detected
  int tape = 300;

  // go forward... unless the user tells you to do something else

      char choice = Serial.read();
      Serial.println(choice);

      if(choice>='0'){
       current = choice;
      }
          if(current == '0'){
      //      turn left
              backMotor->setSpeed(100);
              leftMotor->setSpeed(100);
              rightMotor->setSpeed(100);
              frontMotor->run(RELEASE);
              leftMotor->run(FORWARD);
              rightMotor->run(BACKWARD);
              palletMotor->run(RELEASE);
           }
           else if(current == '1'){
           }
           else if(current == '7'){
      //      move pallet jack down
           }

           else if(current =='9'){ //Line follow go forward
              backMotor->setSpeed(50);
              leftMotor->setSpeed(50);
              rightMotor->setSpeed(50);

              if(sLeft >= tape && sRight >= tape){
                  backMotor->run(RELEASE);
                  leftMotor->run(RELEASE);
                  rightMotor->run(RELEASE);
              }

              else if(sLeft >= tape){  // turn left
                  backMotor->run(RELEASE);
                  leftMotor->run(BACKWARD);
                  rightMotor->run(FORWARD);
              }

              else if(sRight >= tape){
                  backMotor->run(RELEASE);
                  leftMotor->run(FORWARD);
                  rightMotor->run(BACKWARD);
              }

              else{
                  backMotor->run(RELEASE);
                  leftMotor->run(FORWARD);
                  rightMotor->run(FORWARD);
              }

      }

      else{
  //      without any input stay put
          backMotor->run(RELEASE);
          leftMotor->run(RELEASE);
          rightMotor->run(RELEASE);
          palletMotor->run(RELEASE);
      }
      delay(10);
}
```

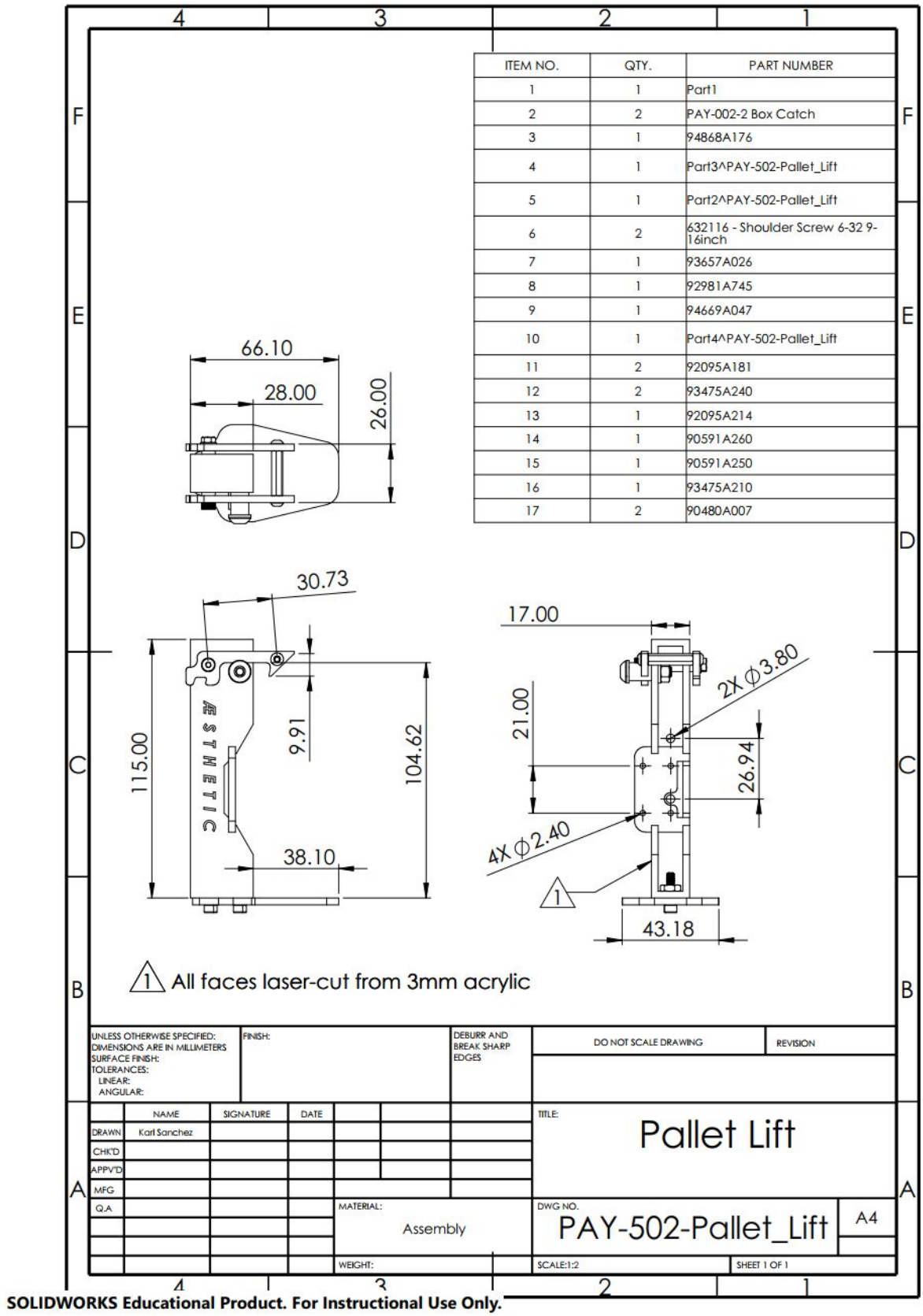
Payload loop: Pallet Alignment

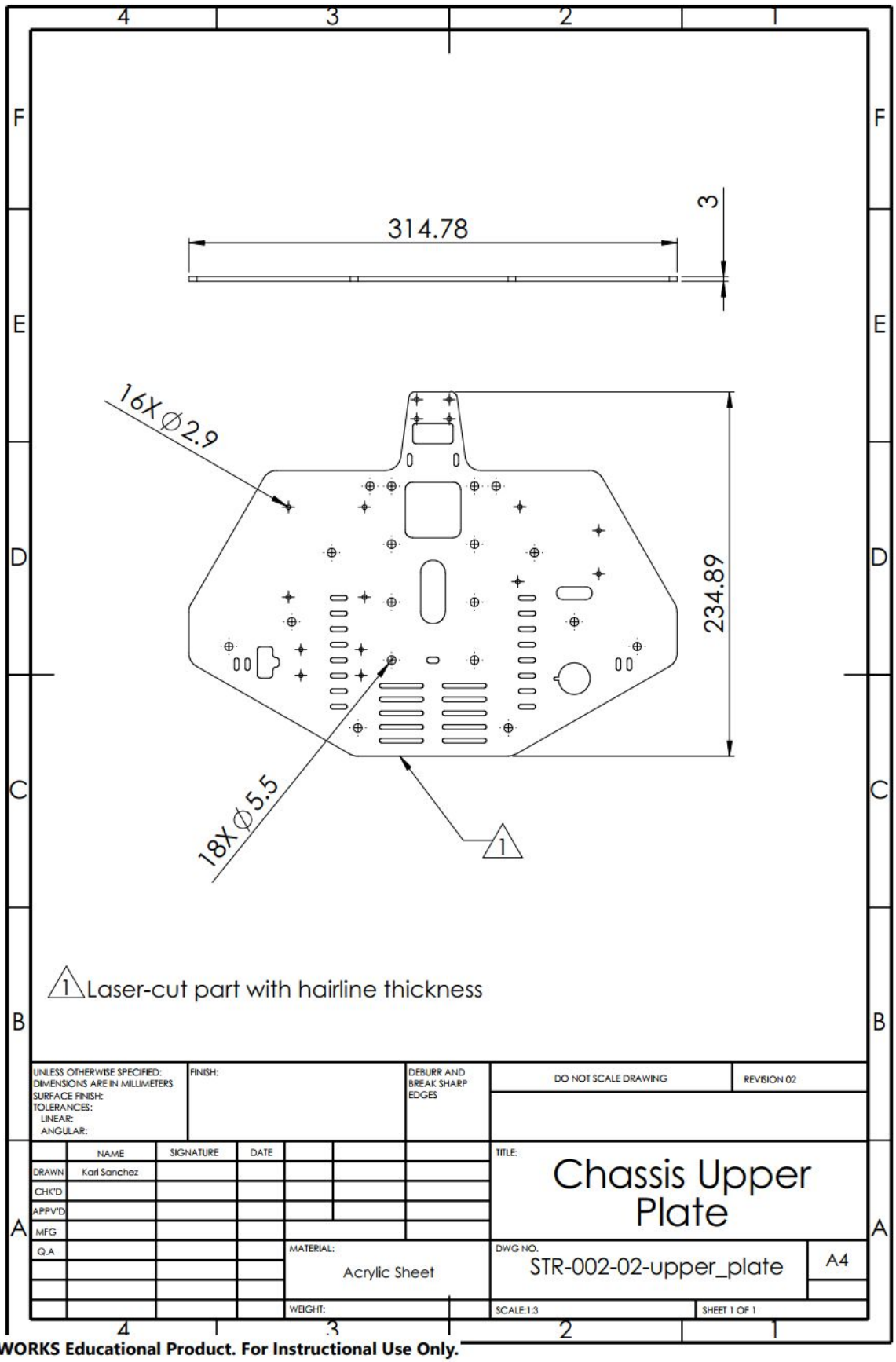
```python
# Authors:
#    Aisling Pigott
#    Kunal Shinde
#    Arth Beladiya
#    Karl Sanchez
# Aligns robot with QR codes

from imutils.video import VideoStream
from pyzbar import pyzbar
import imutils
import time
import cv2
import struct
import serial

# change this variable for alignment tolerance.
alignTol = 20

# initialize Arduino connection
arduino = serial.Serial('/dev/ttyUSB0',9600,timeout=.1)
arduino.flushInput()

# initialize the video stream and allow the camera to warm up
print("[INFO] starting video stream...")
vs = VideoStream(usePiCamera=True).start()

# give some time for video stream and arduino serial comm. to initialize
time.sleep(3)

while True:
    # grab frame from threaded video stream and resize it to max width of 400 px
    frame = vs.read()
    frame = imutils.resize(frame, width=400)

    # find the barcodes in the frame
    barcodes = pyzbar.decode(frame)

    # loop over detected barcodes
    for barcode in barcodes:
        # extract bounding box location
        (x, y, w, h) = barcode.rect

        # convert barcode data to a sting
        barcodeData = barcode.data.decode("utf-8")
        # check that barcode is the one you want to align with
        barcodeXCenter = x + w/2 #horz center of barcode used for alignment
        barcodeYCenter = y + h/2 #vert center of barcode used for alignment

        # move the robot to center over the axis parallel to the pallet rack
        if(barcodeXCenter < (200-alignTol)):
            instructions = "traverse right!\n"

        elif(barcodeXCenter > (200+alignTol)):
            instructions = "traverse left!\n"

        else: # when the robot is horizontally centered

            if(barcodeYCenter < (200-alignTol)):
                instructions = "move pallet jack up!"
            elif(barcodeYCenter > (200+alignTol)):
                instructions = "move pallet jack down!"
            else: # when the robot is centered and the pallet jack is centered
                if(w*h < 40000): # when robot hits some specified distance away
                    instructions = "move forward!\n"
                else: # when at a predefined start point you can start the palle
                    runPalletJack()

        # convert instructions to integer value
        arduino.write(struct.pack('>B',instructions))

    #show output frame
    cv2.imshow("Barcode Scanner", frame)

print("[INFO] cleaning up...")
cv2.destroyAllWindows()
vs.stop()
```

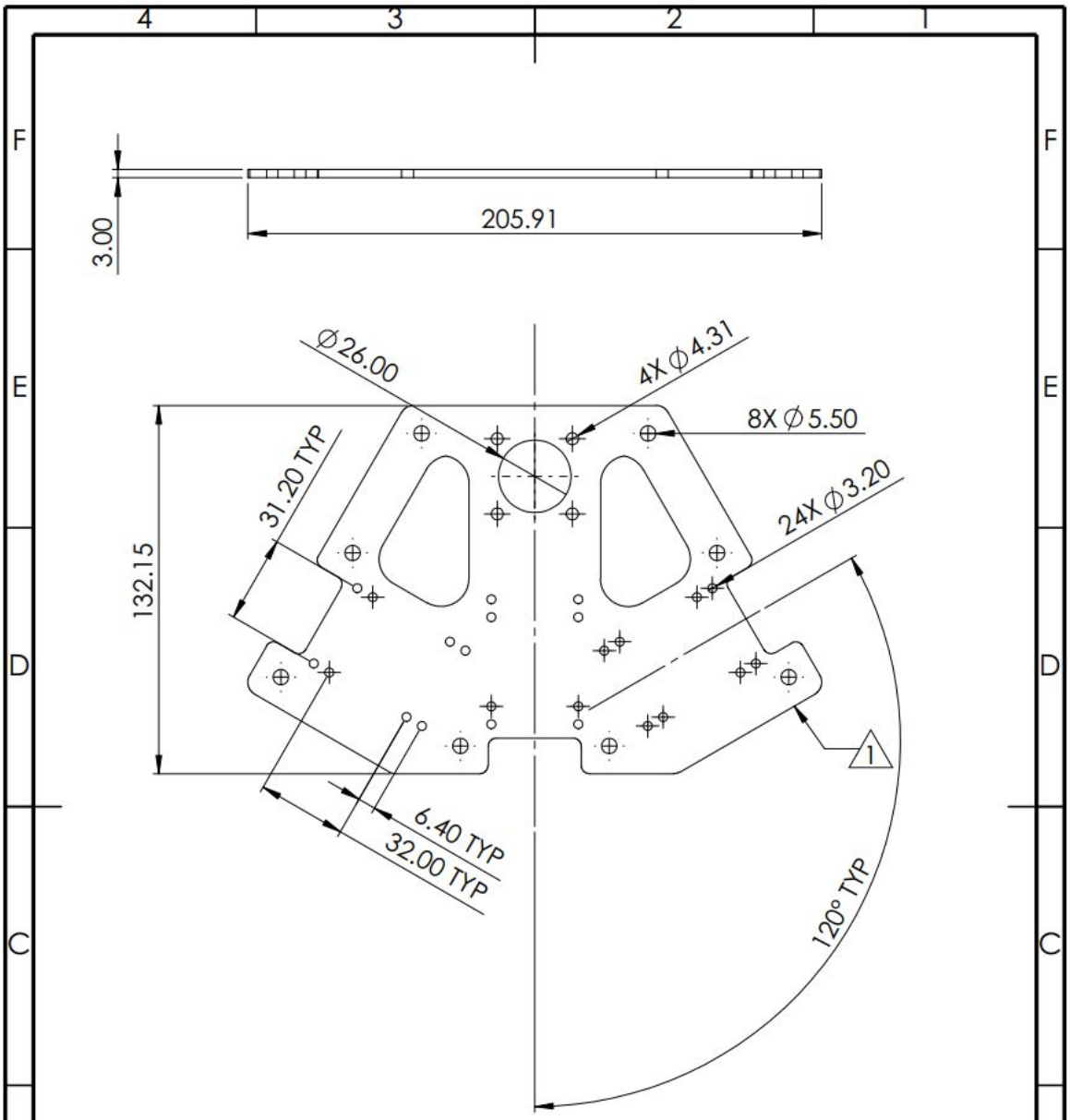
Payload Loop: Pallet Extraction

```python
# Authors:
#    Aisling Pigott
#    Kunal Shinde
#    Arth Beladiya
#    Karl Sanchez
# Sends time based extraction instructions to the Arduino via groundPi

import time
import serial

# assume that groundPi serial connection is initialized

def runPalletJack():
    # start point is some specified distance from the pallet rack
    groundPi.write(4) # move forward
    # groundPi writes this to Arduino
    time.sleep(0.5) # keep moving forward
    groundPi.write(6) # move pallet jack up
    # groundPi writes this to Arduino
    time.sleep(1)
    groundPi.write(5) # move backwards
    # groundPi writes this to Arduino
    time.sleep(1.5)
    groundPi.write(28) # mission success signal

```

Selected Mechanical Drawings







Bill of Materials

Subsystem	Item Description	Vendor	P/N	Model	Units	Unit Price	Total Price
CDH	Raspberry Pi	Derek	-	Raspberry Pi 3B	2	0	0
CDH	Microcontroller	Derek	-	Arduino Redboard	1	0	0
CDH	Pi Camera	Derek	-	V2			0
CDH	Line Sensor	Derek	-	QRB1113	2	0	0
CDH	USB Battery	Karl	-	USB Battery	1	0	0
CHAS	Frame Material	McGuckin	-	0.118" Acrylic	1	8	8
CHAS	Hex Standoff	McMaster	95947A563	Aluminum Female Threaded Hex Standoff	8	2.71	21.68
DRIVE	Omni wheels	Robot Source	RB-Nex-08	100mm Omni Wheel	3	20.16	60.48
DRIVE	Wheel Hub	SuperDroid Robots	TD-094-006	6mm Set Screw Hub	3	12	36
DRIVE	Voltage Regulator	Pololu	S18V20ALV	4-12V Step-Up/Step-Down Voltage Regulator	1	16.99	16.99
DRIVE	Drivetrain Motor	Jameco	253500	DC Motor with Gearhead 12VDC 74Ma	3	0	0
DRIVE	Drivetrain Motor Mount	Pololu	1084	Pololu Stamped Aluminum L-Bracket Pair for 37D mm Metal Gearmotors	3	0	0
DRIVE	Drivetrain Motor Driver	AdaFruit	1438	Motor/Stepper/Servo Shield for Arduino v2 Kit - v2.3	1	19.95	19.95
DRIVE	Battery	Derek	-	Turnigy 4000mAh 4S 30C Lipo Pack w/XT-60	1	0	0
Materials	3D Print Spool	ITLL	-	-	1	10	10
Materials	Kunal Orders	-	-	-	1	10	10
Materials	Karl Orders	-	-	-	1	40	40
PAY	Lead Screw	Servo City	3501-0606-0350	350mm 6mm Lead Screw	1	7.99	7.99
PAY	Bearings	Servo City	535220	6mm ID x 12mm OD Flanged Ball Bearing (2 Pack)	1	3.39	3.39
PAY	Lead Screw nut	Servo City	545308	6mm Lead Screw Nut for Open X-Rail	1	12.99	12.99
PAY	Guide Rail	Servo City	565126	13.50" X-Rail	1	5.49	5.49
PAY	Vertical motor	Servo City	638354	170 RPM Econ Gear Motor	1	14.99	14.99
PAY	Shaft Couplers	Servo City	625081	Clamping Shaft Couplers Bore 4mm to 6mm	1	4.99	4.99
PAY	Hub	Servo City	545682	25mm Bore, Face Tapped Clamping Hub, 1.50" Pattern	1	5.99	5.99
PAY	Payload Hardware	Servo City	-	Hardware	1	22	22
				Total Cost			**300.93**