# Human Target Tracking System

## System Design

**Ahmad Barqawi**

**Abdullah Abu-Shamleh**

**Date: 9/29/2022**

## Problem Statement

As the use of computers and cameras increases by the day, the need to automate applications this pair can do increases as well. In the application of surveillance, the fast-growing task of automated human tracking is becoming essential. However, due to it being challenging to track the generally changing human appearance, having a system that can do so has enormous potential in a lot of fields.

## Introduction

These days, the need of a surveillance system in field like public transport, military, urban planning, and healthcare has magnified the attention towards developing such systems and designs. Alongside, the exponential growth in the fields of computing and artificial intelligence serves as a huge help in designing portable systems that do the task of tracking efficiently. Artificial intelligence techniques for image detection are the building blocks for many recent implementations of tracking, and it is indeed what will be used in this project as well.

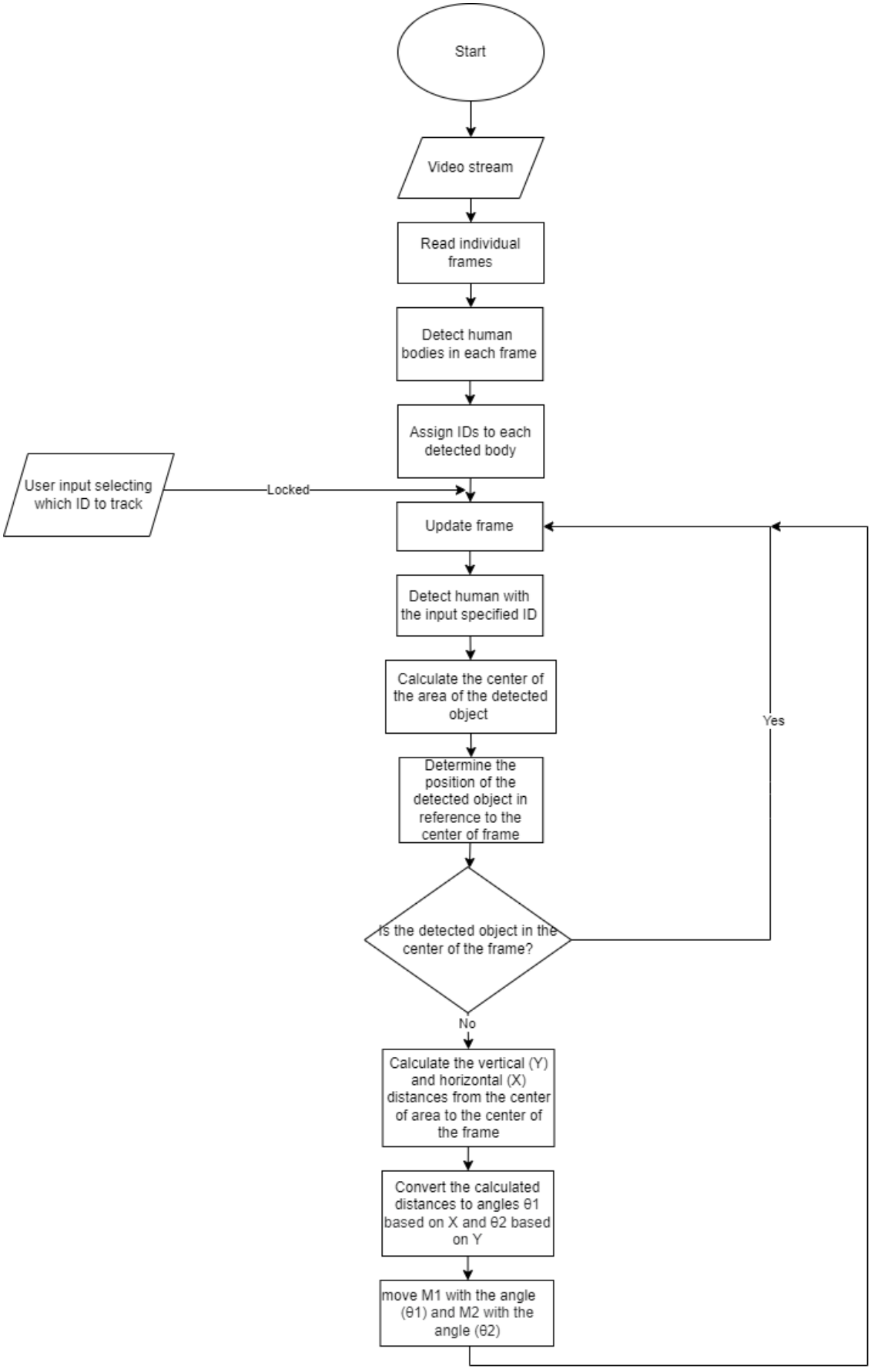
## Project Description

This system will be designed containing multiple parts, both software and hardware, to achieve the task of specific target human tracking. The system will read an input video stream of the operating area and proceed to utilize its' parts alongside user input to track a specific human amongst many other in such area.

The design will include an integrated robotic system responsible for performing the main movements to the camera, thus increasing the operating area specified originally by the range of the camera itself. The design will also use a convolutional neural network responsible for detecting human bodies, alongside an estimation algorithm responsible for keeping track of detections in consecutive frames. All of this will be processed on a computer taking advantage of the immense advancements in portable computing power targeted straight towards artificial intelligence and its' applications.

To conclude, a design like this will be able to produce an efficient high frame rate take on human body tracking by integrating the output of the tracking implementation and controlling the robotic system to enable the camera to do the tracking, all of which can be used for other future projects in the many fields that can take advantage of it.

## Flowchart


Start
Video stream
Read individual frames
Detect human bodies in each frame
Assign IDs to each detected body
User input selecting which ID to track
Locked
Update frame
Detect human with the input specified ID
Calculate the center of the area of the detected object
Determine the position of the detected object in reference to the center of frame
Is the detected object in the center of the frame?
Yes
No
Calculate the vertical (Y) and horizontal (X) distances from the center of area to the center of the frame
Convert the calculated distances to angles θ1 based on X and θ2 based on Y
move M1 with the angle (θ1) and M2 with the angle (θ2)

## Pseudocode

Read video stream from camera

Initialize (TargetID) to NULL

Initialize C = center of area of input frame

Loop1:

Read frame from video stream

Detect human bodies in frame using a deep learning algorithm

For each detected body:

Set the 4 corners (cr1,cr2,cr3,cr4) equal to the deep learning algorithm output

Draw a rectangle with corners (cr1,cr2,cr3,cr4)

Assign a specific ID to the rectangle

Read user input as mouse click on body rectangle desired to be tracked

Set TargetID = The ID of rectangle selected from user input

If TargetID is equal to NULL:

Goto Loop1

Loop2:

Read frame from video stream

Detect human body specified with TargetID

Update rectangle corner associated with targeted body

Calculate OC = Center of rectangle with corners (cr1,cr2,cr3,cr4), and ID = TargetID

If OC = C:

Goto Loop2

else:

Calculate the horizontal distance from the center of the rectangle to the center of frame (X)=$X_c$-$X_{oc}$

Calculate the vertical distance from the center of the rectangle to the center of frame (Y)=$Y_c$-$Y_{oc}$

Calculate the angle needed to center the rectangle horizontally Theta1 from X

Calculate the angle needed to center the rectangle vertically Theta2 from Y

Move motor M1 with angle Theta1

Move motor M2 with angle Theta2

Goto Loop2

## Design Requirements and Considerations

To be able to choose the required components later, the design requirements and considerations shall be stated.

1. **Computer:**

   A small form factor computer with CUDA enabled GPU compute power to handle real time data collection, run neural networks to detect bodies and processing.

   Computer requirements:

   a. Compute capability (as per NVIDIA standard) > 7.1.
   b. Multiple USB Type A ports.
   c. Weight < 200 gram.
   d. Runs on 5V and less than 3A.
   e. Maximum size of 120mm x 120mm x 40mm.

2. **Payload:**

   A USB camera will be used to provide the real time video stream of the operating area and an attached laser module will be attached to it and pointing at the tracked target giving a real-world indication of the system's performance.

   Camera requirements:

   a. Compatible with the USB ports on the computer.
   b. Live video resolution of 1280 x 720 at up to 30 FPS.
   c. Can produce videos of humans at the specified resolution from a range of 2 to 5 meters.
   d. Weighs less than or equal to 165 grams.
   e. Maximum size of 10mm x 5mm x 5mm.

   The laser module needs to be able to run on 5V or less, have a range of 1 meter to 3 meters and weigh less than or equal to 5 grams.

3. **Robotic System:**

   A two degrees of freedom (pan & tilt) robotic system is the considered design for moving the payload of the camera and laser. The links of the robotic system will be moved by rotational uniaxial joints. The motors need to be able to produce smooth movements to specific angles on the rotational range, work on 5V or less on less than 3A, have enough torque to move the system and weigh less than or equal to 50 grams each.

   The range of motion for the motors are required to be between -90° and 90° for panning and between -30° and 60° for tilting. The design of the robot links is required to allow the links to move according to the specified range of motion.

   The robotic system will be stationary and placed on a static base which in turn will be placed on a horizontal or sloped surface (max slope of 60%).

4. **Power Source:**

   A power bank is required to provide power to the system. This bank needs to be able to power the computer, payload and the 2 motors responsible for the robotics movements. The bank is required to provide at least two ports of 5V 3A each (one for the computer and an extra one in case the motors need external power) and be able to power the system for an operational period of at least 3 hours.

## Design and Component Selection

1. **Computer:**
   The Jetson Xavier NX Developer Kit is the computer to be used in the system. The small yet powerful nature of the Jetson Xavier NX allows it to run multiple neural networks.

*Figure 1: Jetson Xavier NX Developer Kit*

*Table 1: Jetson Xavier NX Developer Kit Specifications*

| **Technical Specifications** [1] | |
|---|---|
| **GPU** | NVIDIA Volta architecture with 384 NVIDIA CUDA® cores and 48 Tensor cores |
| **CPU** | 6-core NVIDIA Carmel Arm®v8.2 64-bit CPU |
| **Memory** | 8 GB 128-bit LPDDR4x |
| **Storage** | microSD |
| **Compute Capability** | 7.2 |
| **Video Encode** | 2x 4K @ 30 \| 6x 1080p @ 60 \| 14x 1080p @ 30 (H.265/H.264) |
| **Video Decode** | 2x 4K @ 60 \| 4x 4K @ 30 \| 12x 1080p @ 60 \| 32x 1080p @ 30 (H.265)<br>2x 4K @ 30 \| 6x 1080p @ 60 \| 16x 1080p @ 30 (H.264) |
| **Camera** | 2x MIPI CSI-2 DPHY lanes |
| **Connectivity** | Gigabit Ethernet, M.2 Key E (WiFi/BT included), M.2 Key M (NVMe) |
| **Display** | HDMI and display port |
| **USB** | 4x USB 3.1, USB 2.0 Micro-B |
| **Others** | GPIO, $I^2C$, $I^2S$, SPI, UART |
| **Dimensions** | 10.3cm x 9.05cm x 3.46cm |
| **Weight** | 172 g |
| **Power draw** | 10W in default mode @ 5V (2A) |

2. **Payload:**

The USB camera that will be used is the HP HD-3100 webcam and it will be connected to the Jetson Xavier NX via its' USB cable.

*Figure 2: HP HD-3100 Webcam*

*Table 2: HP HD-3100 Webcam Specifications*

| **Technical Specifications** [2] [3] | |
|---|---|
| **Maximum Video Resolution** | 1280 x 720 pixels |
| **Maximum frame rate** | 30 FPS |
| **Focus method** | range approximately 12 inches (30cm) to 10 feet (300cm) |
| **USB Certification** | USB 2.0 high-speed certified. |
| **Power Consumption** | 5V at 500mA maximum |
| **Weight** | 100 g |
| **Dimensions** | 7.4cm x 3cm x 3.5cm |

The laser module to be used is the KY-008 Laser Transmitter and it will be placed on top of the camera.

*Figure 3: KY-008 Laser Transmitter*

*Table 3: KY-008 Laser Transmitter Specifications*

| **Technical Specifications** [4] | |
|---|---|
| **Voltage** | 5 V |
| **Operating Current** | < 40 mA |
| **Weight** | 3 g |
| **Dimensions** | 2.7cm x 1.5cm x 0.9cm |

## 3. Robotic System:

The two degrees of freedom robotic system design which will be used consists of many aluminum alloy parts like multi-function brackets, U-type brackets, motors and a bunch of bearings, screws, and horns.

The base of the robot will be comprised of 2 U-type beam brackets and the first motor will be placed on top of the base in housed in a multi-function bracket. The first motor will control the movement of the multi-function bracket (pan), where the second motor will be in housed, creating the panning angle Θ. The second motor will control the movement of the U-type bracket (tilt), where the camera and laser will be attached, creating the tilting angle α. The range of motion in the design will be constrained as per requirements.

a. The U-Shaped Beam Brackets used:

*Figure 4: U-Shaped Beam Bracket*

*Table 4: U-Shaped Beam Bracket Specifications*

| **Technical Specifications** [5] | |
|---|---|
| **Weight** | 50 g |
| **Dimensions** | 9cm x 4.2cm x 2.5cm |

b. The Multi-Function Brackets used:

*Figure 5: Multi-Function Bracket*

*Table 5: Multi-Function Bracket Specifications*

| **Technical Specifications** [6] | |
|---|---|
| **Weight** | 16 g |
| **Dimensions** | 5.8cm x 3.7cm x 2.7cm |

c. The Long U-Shaped Bracket used:

*Figure 6: Long U-Shaped Bracket*

*Table 6: Long U-Shaped Bracket Specifications*

| **Technical Specifications** [7] | |
|---|---|
| **Weight** | 20 g |
| **Dimensions** | 5.6cm x 2.5cm x 6.4cm |

The motors that will be used to control the robotic system are DC servo motors. This comes since they are small and powerful. Servo motors also come with control circuitry that allow almost perfect repeatability of motion to specific precise angles. However, to be able to choose a servo motor model, the maximum torque must be calculated. Two test cases for each motor will be considered, the first on a horizontal surface and the second on a 60% slope. (All forces are divided by the gravitational acceleration constant to find the torque in kg.cm).

First motor on a horizontal surface:

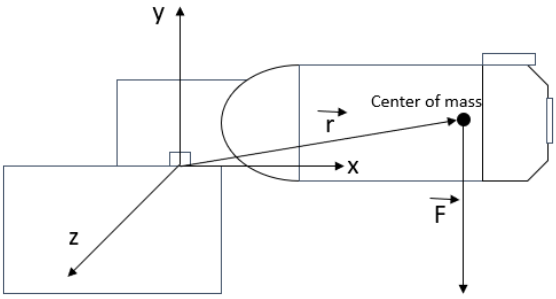



*Figure 7: The First Servo Motor with Its' Axes on a horizontal surface, It Rotates Around its' Y Axis*

$\vec{r}$= 5.375cos(α)cos(Θ)+2cos(Θ))i + (5.375sin(α)+1)j + (5.375cos(α)sin(Θ)+sin(Θ))k cm

$\vec{F}$ = -0.26j kg

$\vec{M}$= $\vec{r}$ x $\vec{F}$

$\vec{M}$= 0.26(5.375cos(α)sin(Θ)+sin(Θ))i + 0j - 0.26(5.375cos(α)cos(Θ)+2cos(Θ))k kg.cm

The first motor will rotate around the y axis so the maximum absolute value of the j component in the moment vector is the torque value that helps in choosing the motor model. Since the value is equal to zero then the motor will not hold any torque and selection is not constrained.

Second motor on a horizontal surface:

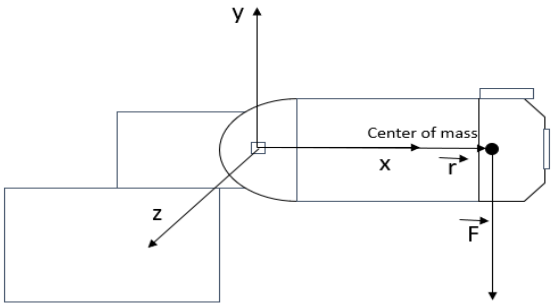



*Figure 8: The Second Servo Motor with Its' Axes on a horizontal surface, It Rotates Around its' Z Axis*

$\vec{r}$ = 5.375cos(α)i + 5.375sin(α)j + 2.5k  cm

$\vec{F}$ = -0.19j  kg

$\vec{M}$= $\vec{r}$ x $\vec{F}$

$\vec{M}$= 0.342i + 0j - 1.02125cos(α)k  kg.cm

The second motor will rotate around the z axis so the maximum absolute value of the z component in the moment vector is the torque value that helps in choosing the motor model. Since the value is equal to 1.02125 kg.cm at α=0° then the motor chosen needs to have torque higher than this value.

First motor on a 60% slope:

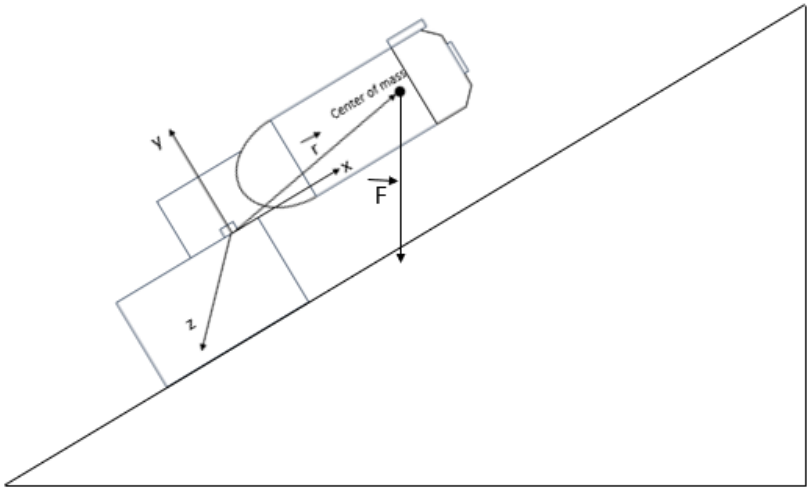



*Figure 9: The First Servo Motor with Its' Axes On a 60% Slope, It Rotates Around its' Y Axis*

$\vec{r}$= 5.375cos(α)cos(Θ)+2cos(Θ))i + (5.375sin(α)+1)j + (5.375cos(α)sin(Θ)+sin(Θ))k cm

$\vec{F}$ = -0.26sin(31°)i - 0.26cos(31°)j kg

$\vec{M}$= $\vec{r}$ x $\vec{F}$

$\vec{M}$= 0.22286(5.375cos(α)sin(Θ)+sin(Θ))i - 0.1339(5.375cos(α)sin(Θ)+sin(Θ))j - (0.22286(5.375cos(α)cos(Θ)+2cos(Θ))+0.1339(5.375sin(α)+1))k kg.cm

The first motor will rotate around the y axis so the maximum absolute value of the j component in the moment vector is the torque value that helps in choosing the motor model. Since the value is equal to 0.854 kg.cm at α=0° and Θ=-90°,90° then the motor chosen needs to have torque higher than this value.

Second motor on a 60% slope:

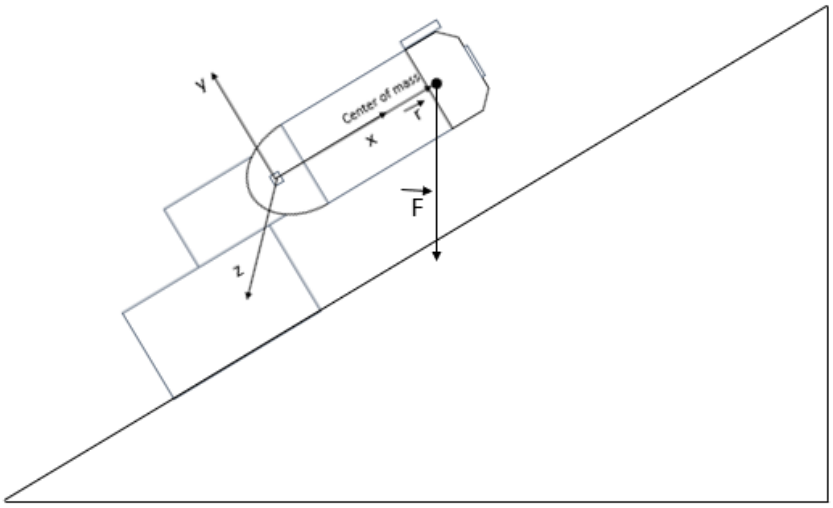



*Figure 10: The Second Servo Motor with Its' Axes On a 60% Slope, It Rotates Around its' Z Axis*

$\vec{r}$ = 5.375cos(α)i + 5.375sin(α)j + 2.5k cm

$\vec{F}$ = -(0.19(sin(31°)cos(Θ)cos(α)+cos(31°)sin(α))i + (sin(31°)cos(Θ)sin(α)-cos(31°)cos(α)) + (sin(31°)sin(Θ))k kg

$\vec{M}$= $\vec{r}$ x $\vec{F}$

$\vec{M}$= (0.525982sin$^2$(α)-0.342(0.51503sin(α)cos(Θ)-0.85716cos(α)))i + (0.342(-0.51503cos(α)cos(Θ)-0.85716sin(α))-0.525982cos(α)sin(α))j + (1.051965sin(α)cos(Θ)-0.875382cos$^2$(α)+0.875382sin$^2$(α))k kg.cm

The second motor will rotate around the z axis so the maximum absolute value of the z component in the moment vector is the torque value that helps in choosing the motor model. Since the value is equal to 1.021 kg.cm at Θ=0° and α=-15.03° then the motor chosen needs to have torque higher than this value.

From the values calculated above and considering that the same model of motor will be used for both motors, the maximum value of 1.02125 kg.cm will be the considered value of the stall torque for the motors in the design.

As for the motors speed, motors with a speed value of 0.15sec/60° are the most used in designs for tracking. [8]

d. MG996R Servo Motor:

*Figure 11: MG996R Servo Motor*

*Table 7: MG996R Servo Motor Specifications*

| **Technical Specifications** [9] | |
|---|---|
| **Stall torque** | 9.4kg/cm (4.8v); 11kg/cm (6v) |
| **Operating speed** | 0.15sec/60degree |
| **Operating voltage** | 4.8~ 6.6V |
| **Gear Type** | Metal gear |
| **No load operating current draw** | 170mA |
| **Current draw @ 1.02125 Kg.cm** | 300mA (Found by using the linear equation between current and torque with stall torque equaling 1400mA) |
| **Weight** | 55 g |
| **Dimensions** | 4cm x 4.2cm x 2cm |

Since the Jetson Xavier NX cannot drive servo motors, an Arduino UNO R3 is going to be used to assist in driving the motors. [10]

*Figure 12: Arduino UNO R3*

*Table 8: Arduino UNO R3 Specifications*

| **Technical Specifications** [11] | |
|---|---|
| **Main Processor** | ATmega328P 16 MHz |
| **USB-Serial Processor** | ATmega16U2 16 MHz |
| **I/O Voltage** | 5V |
| **Input voltage (nominal)** | 7-12V |
| **Maximum current draw** [12] | 200mA |
| **Weight** | 25 g |
| **Dimensions** | 8cm x 5.5cm x 2.4cm |

### *4.* Power Source:

*Table 9: Current Draw Needed for the System*

| Component | Quantity | Maximum current draw |
|---|---|---|
| **Jetson Xavier NX** | 1 | 2A |
| **MG996R servo motor** | 2 | 0.3A (total = 0.6) |
| **Arduino UNO R3** | 1 | 0.2A (supplied from the Xavier) |
| **HP HD 3100 webcam** | 1 | 0.5A (supplied form the Xavier) |
| **KY-008 Laser Transmitter** | 1 | 0.04A (supplied from the Arduino) |
| **Total maximum current draw from the power source** | 2.6A | |

To provide the needed current for the Jetson Xavier NX with the connected Arduino, as well as the needed current for the motors to operate, a portable power bank is going to be used. This power bank will have 2 USB outputs each providing 5V 3A, one going into the Jetson Xavier NX and the other into the 2 servo motors.

The Charmast Portable Charger that will be used:

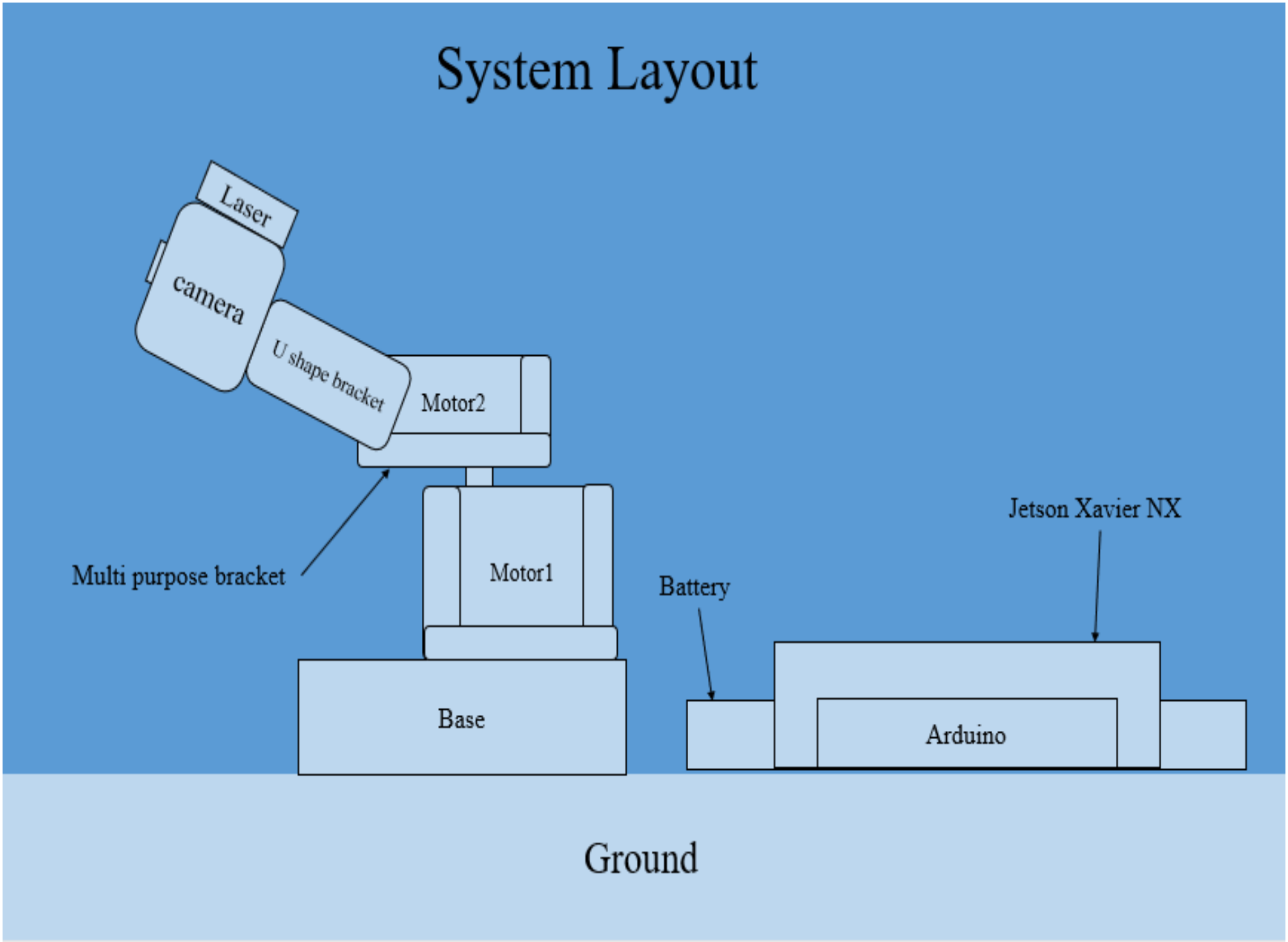

*Figure 13: Charmast Portable Charger*

*Table 10: Charmast Portable Charger Specifications*

| **Technical Specifications** [13] | |
|---|---|
| **Model** | W1056 |
| **Rated Energy** | 38.48Wh |
| **Capacity** | 10400mAh |
| **Input Ports** | Micro input 5V-2A; USB C input 5V-3A |
| **Output Ports** | Total (USB 1/2 + USB C output )5V 3A (15W) max |
| **Weight** | 228g |
| **Dimensions** | 14.4cm x 6.7cm x 1.52cm |

*Table 11: Components Summary*

| Component | Dimensions | Weight | Quantity |
|---|---|---|---|
| **HP Webcam HD-3100** | 7.4cm x 3cm x 3.5cm | 100 grams | 1 |
| **KY-008 Laser Transmitter** | 2.7cm x 1.5cm x 0.9cm | 3 grams | 1 |
| **U-shaped beam bracket** | 9cm x 4.2cm x 2.5cm | 50 grams | 2 |
| **Multi-Function Bracket** | 5.8cm x 3.7cm x 2.7cm | 16 grams | 2 |
| **Long U-Shaped Bracket** | 5.6cm x 2.5cm x 6.4cm | 20 grams | 1 |
| **MG996R Servo Motor** | 4cm x 4.2cm x 2cm | 55 grams | 2 |
| **Jetson Xavier NX Developer Kit** | 10.3cm x 9.05cm x 3.46cm | 172 grams | 1 |
| **Arduino UNO R3** | 8cm x 5.5cm x 2.4cm | 25 grams | 1 |
| **Charmast Portable Charger** | 14.4cm x 6.7cm x 1.52cm | 228 grams | 1 |
| | | Total = 759 grams | |


System Layout
Laser
camera
U shape bracket
Motor2
Multi purpose bracket
Motor1
Base
Battery
Jetson Xavier NX
Arduino
Ground

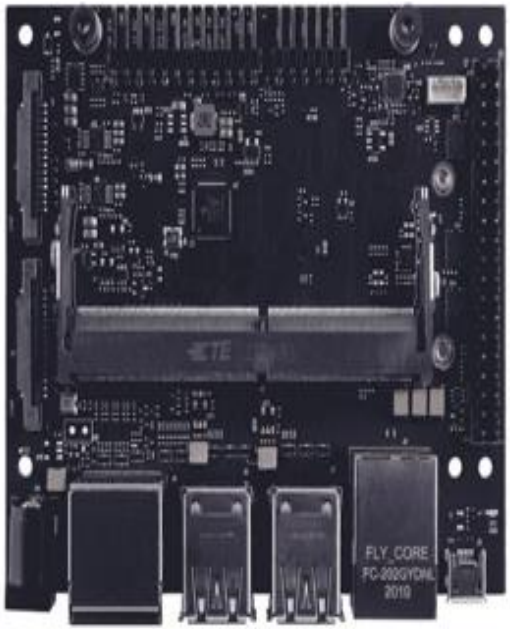
# Schematic Diagram

Charmast Portable
Charger
with 10,400 mAh

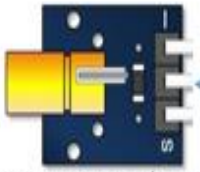
Model #:W1056

USB A-Power cable

Jetson Xavier NX kit



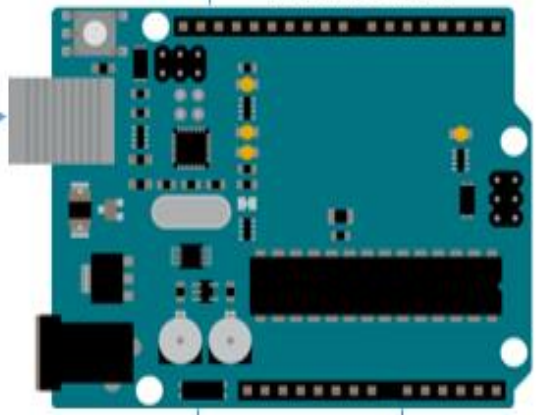
USB A-USB2 Micro B



Male/Female wires (2 wires)

KY-008 Laser / 40mA

Arduino UNO R3



2 Male/Male jumper wires

2 Male/Male jumper wires

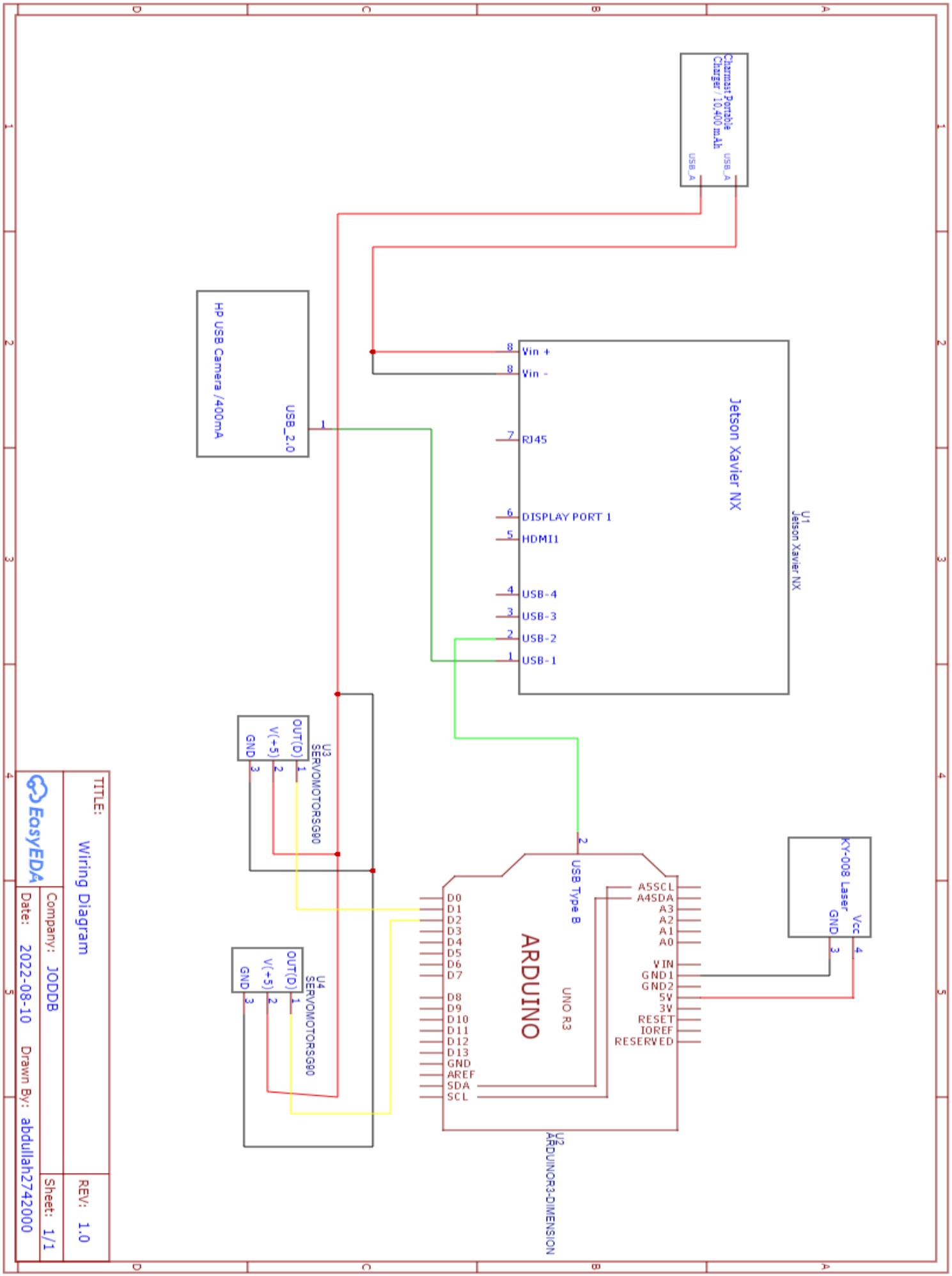
USB A-USB B

From camera-USB A

HP USB Webcam 3100/400mA
Part #:BK356AA#ABA


Servo motor MG996R

Servo motor MG996R


Charmast Portable Charger / 10,400 mAh
USB_A
USB_A
HP USB Camera /400mA
USB_2.0
1
8 Vin +
8 Vin -
7 RJ45
6 DISPLAY PORT 1
5 HDMI1
4 USB-4
3 USB-3
2 USB-2
1 USB-1
Jetson Xavier NX
U1
Jetson Xavier NX
OUT(D)
V(~5)
GND
U3
SERVOMOTORSG90
OUT(D)
V(~5)
GND
U4
SERVOMOTORSG90
2
USB Type B
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
GND
AREF
SDA
SCL
ARDUINO
UNO R3
A5SCL
A4SDA
A3
A2
A1
A0
VIN
GND1
GND2
5V
3V
RESET
IOREF
RESERVED
U2
ARDUINOR3-DIMENSION
KY-008 Laser
Vcc
GND
3
4
TITLE:
Wiring Diagram
EasyEDA
Company: JODDB
Date: 2022-08-10
Drawn By: abdullah2742000
Sheet: 1/1
REV: 1.0

## Software Implementation

The software for the project has been divided into 2 parts. The first part does the software tracking by detecting human bodies and assigning IDs to each detection, enables the choosing of a specific human body to be tracked by the robotic system and sends that body's coordinates to the Arduino. This part is run on the Jetson Xavier NX.

The second part does the hardware tracking of a specific single human body by controlling the motors and moving the camera until that body is centered in the frame. This part is run on the Arduino UNO R3.

### Part I: Jetson Xavier NX

The tracker which will be used in this project is a custom MOT (Multiple Object Tracker) implementation called FastMOT [14]. This tracker implements detection using YOLO, Deep SORT + OSNet ReID, KLT tracking and camera motion compensation. It also achieves great performance especially on Jetson devices by running the detection models every N frames while KLT fills the gaps efficiently.

The Jetson Xavier NX being used is running the latest JetPack 5.0.2 version which already includes CUDA 11.4, TensorRT 8.4.1, cuDNN 8.4.1 and Python 3.8.10.

To begin with, make sure the Jetson Xavier NX operating system is up to date by running these commands:

```
sudo apt-get update

sudo apt upgrade

sudo apt install
```

Secondly, clone into the FastMOT repository on github:

```
git clone https://github.com/GeekAlexis/FastMOT
```

Thirdly, install the tracker by running the script found in the FastMOT directory then download the models to be used:

```
./scripts/install_jetson.sh

./scripts/download_models.sh
```

The installed dependencies from the scripts are the following:

1. NumPy 1.17
2. Scipy 1.5
3. Numba 0.48
4. TensorFlow 1.15.5
5. CuPy 9.2
6. Gdown 4.5.1

Note: The serial library (version 3.5) needs to be installed as well because it will not be found in the scripts:

```
pip3 install pyserial
```

Finally, build the YOLOv4 TensorRT plugin:

```
cd fastmot/plugins
make
```

The final script being used for tracking and communicating with the Arduino will be discussed in main sections:

**Imports**

```
#!/usr/bin/env python3

from pathlib import Path # Already Installed with Python 3.8.10 on Jetson Xavier NX using JetPack 5.0.2
from types import SimpleNamespace # Already Installed with Python 3.8.10 on Jetson Xavier NX using JetPack 5.0.2
import argparse # Already Installed with Python 3.8.10 on Jetson Xavier NX using JetPack 5.0.2
import logging # Already Installed with Python 3.8.10 on Jetson Xavier NX using JetPack 5.0.2
import json # Already Installed with Python 3.8.10 on Jetson Xavier NX using JetPack 5.0.2
import cv2 # Already Installed with Python 3.8.10 on Jetson Xavier NX using JetPack 5.0.2
import time # Already Installed with Python 3.8.10 on Jetson Xavier NX using JetPack 5.0.2
import serial # Installed Manually

import fastmot # Installed Manually and found in the FastMOT directory
import fastmot.models # Installed Manually and found in the FastMOT directory
from fastmot.utils import ConfigDecoder, Profiler # Installed Manually and found in the FastMOT directory
```

**Main function: Setting up the argument parser**

```
def main():
    parser = argparse.ArgumentParser(formatter_class=argparse.RawTextHelpFormatter)
    optional = parser._action_groups.pop()
    required = parser.add_argument_group('required arguments')
    group = parser.add_mutually_exclusive_group()
    required.add_argument('-i', '--input-uri', metavar="URI", required=True, help=
                        'URI to input stream\n'
                        '1) image sequence (e.g. %%06d.jpg)\n'
                        '2) video file (e.g. file.mp4)\n'
                        '3) MIPI CSI camera (e.g. csi://0)\n'
                        '4) USB camera (e.g. /dev/video0)\n'
                        '5) RTSP stream (e.g. rtsp://<user>:<password>@<ip>:<port>/<path>)\n'
                        '6) HTTP stream (e.g. http://<user>:<password>@<ip>:<port>/<path>)\n')
    optional.add_argument('-c', '--config', metavar="FILE",default=Path(__file__).parent / 'cfg' / 'mot.json',help='path to JSON configuration file')
    optional.add_argument('-m', '--mot', action='store_true', help='run multiple object tracker')
    group.add_argument('-q', '--quiet', action='store_true', help='reduce output verbosity')
    group.add_argument('-v', '--verbose', action='store_true', help='increase output verbosity')
    parser._action_groups.append(optional)
    args = parser.parse_args()
```

**Main function: Setting up logging and loading the config file**
**Frame skips and webcam stream resolution are configurable in the config file.**
**Skipped frames between inferencing has been set to 5 and stream resolution has been set to 640 x 480.**

```
    logging.basicConfig(format='%(asctime)s [%(levelname)8s] %(message)s', datefmt='%Y-%m-%d %H:%M:%S') # Setup logging
    logger = logging.getLogger(fastmot.__name__)
    if args.quiet:
        logger.setLevel(logging.WARNING)
    elif args.verbose:
        logger.setLevel(logging.DEBUG)
    else:
        logger.setLevel(logging.INFO)

    # Load config file
    with open(args.config) as cfg_file:
        config = json.load(cfg_file, cls=ConfigDecoder, object_hook=lambda d: SimpleNamespace(**d))
```

**Main function: Defining the needed instances and variables**

```
stream = fastmot.VideoIO(config.resize_to, args.input_uri, None, **vars(config.stream_cfg)) # Defining the video stream from the webcam
ard=serial.Serial(port='/dev/ttyUSB0',baudrate=115200,timeout=0.1) # Defining the serial port used for communicating with the Arduino

mot = fastmot.MOT(config.resize_to, **vars(config.mot_cfg), draw=False) # Defining an instance of the mot tracker
mot.reset(stream.cap_dt) # Resets the tracker
cv2.namedWindow('Video', cv2.WINDOW_AUTOSIZE) # Define the window showing the video stream

logger.info('Starting video capture...')
global tid #Integer variable that holds the target ID of the human body that will be tracked by the robot
global there #Boolean variable that specifies if there is a body being tracked by the robot still in the frame
tid = -1
stream.start_capture()
```

**Main function: Reading a frame from the stream and tracking the human bodies in it then returning coordinates of the tracked bodies boxes**

**Each tracked body has many parameters like a tracking ID, top left and bottom right (tlbr) coordinates in the frame, state, age and label. In this code, only the tracking ID and the tlbr coordinates will be used directly.**

```
try:
    with Profiler('app') as prof:
        while cv2.getWindowProperty('Video', 0) >= 0:
            frame = stream.read() # Reading the frame
            if frame is None:
                break

            if args.mot:
                mot.step(frame) # Running the tracker on the current frame
                boxes=list()
                for track in mot.visible_tracks(): #Looping over tracked body's boxes in the frame
                    tl = track.tlbr[:2] / config.resize_to * stream.resolution #List holding the top left coordinates of the tracked body box
                    br = track.tlbr[2:] / config.resize_to * stream.resolution #List holding the bottom right coordinates of the tracked body box
                    x1=int(tl[0]) # Variable holding the X coordinate of the top left point for the tracked body's box
                    y1=int(tl[1]) # Variable holding the Y coordinate of the top left point for the tracked body's box
                    x2=int(br[0]) # Variable holding the X coordinate of the bottom right point for the tracked body's box
                    y2=int(br[1]) # Variable holding the Y coordinate of the bottom right point for the tracked body's box
                    if x1<0:
                            x1=0
                    elif x1>640:
                            x1=640
                    if x2<0:
                            x2=0
                    elif x2>640:
                            x2=640
                    if y1<0:
                            y1=0
                    elif y1>480:
                            y1=480
                    if y2<0:
                            y2=0
                    elif y2>480:
                            y2=480
                    width=x2-x1 # Variable holding the width for the tracked body's box
                    height=y2-y1 # Variable holding the height for the tracked body's box
                    box=[track.trk_id,x1,y1,x2,y2]
                    boxes.append(box)
                    startp=(x1,y1)
                    endp=(x2,y2)
```

**Main function: Showing either all the tracked bodies in the frame or only the selected body to be tracked by the robot**

```
if there==False: # Show tracking boxes of all the human bodies in the frame
                        frame = cv2.rectangle(frame,startp,endp,(0,255,0),2)
                        cv2.putText(frame,str(track.trk_id),(int(x1+width/2),int(y1+height/2)),cv2.FONT_HERSHEY_SIMPLEX,0.9,(0,255,0),2)
                else: # Show the tracking box of the human body selected to be tracked by the robot
                        if track.trk_id==tid:
                                frame = cv2.rectangle(frame,startp,endp,(0,255,0),2)
                                cv2.putText(frame,str(track.trk_id),(int(x1+width/2),int(y1+height/2)),cv2.FONT_HERSHEY_SIMPLEX,0.9,(0,255,0),2)

        cv2.imshow('Video', frame) # Show the frame in the window
        if cv2.waitKey(1) & 0xFF == 27: # Break out of the tracking loop if the "Esc" button is pressed
            break
```

**Main function: Sending the coordinates of the selected human body to be tracked by the robot to the Arduino**

**In the case of a body getting clicked on in the webcam stream window, that body's tracking ID will be stored in the tid variable and accordingly it's coordinates each frame will be sent to the Arduino to perform the robot tracking.The sent coordinates are formatted as a string containing the centroid coordinates of the tracked body's box alongside the tracked body's box height.**

```
cv2.setMouseCallback('Video', click_event,boxes) # Calling the click_event_boxes function that returns the clicked body's tracking ID in the variable tid (Defined later on in the code)
        there=False
        if tid != -1:
                for track in mot.visible_tracks():
                        if track.trk_id==tid:
                                tl = track.tlbr[:2] / config.resize_to * stream.resolution
                                br = track.tlbr[2:] / config.resize_to * stream.resolution
                                x1=int(tl[0])
                                y1=int(tl[1])
                                x2=int(br[0])
                                y2=int(br[1])
                                if x1<0:
                                        x1=0
                                if x1>640:
                                        x1=640
                                if x2<0:
                                        x2=0
                                if x2>640:
                                        x2=640
                                if y1<0:
                                        y1=0
                                if y1>480:
                                        y1=480
                                if y2<0:
                                        y2=0
                                if y2>480:
                                        y2=480
                                width=x2-x1 # Variable holding the width for the tracked body's box
                                height=y2-y1 # Variable holding the height for the tracked body's box
                                string='X{0:d}Y{1:d}H{2:d}'.format((x1+width//2),(y1+height//2),height) #Define the string containing the centre of the selected human body box to be tracked by the robot
                                ard.write(string.encode('utf-8')) # Sending the coordinated to the arduino
                                there=True
                                break
                        else:
                                there=False
        if there == False or tid == -1: # Send to the arduino to stop the robot if the selected human body is no longer in the frame
            string='X320Y240' # Sending the coordinates of the centroid for the frame which indicates to stop tracking (moving) with the robot
            ard.write(string.encode('utf-8'))
    finally: # Release the webcam stream and destroy all windows generated by the code after leaving the tracking loop
        stream.release()
        cv2.destroyAllWindows()
```

**Main function: Finding and logging the timing statistics**

```
# Timing statistics
    if args.mot:
        avg_fps = round(mot.frame_count / prof.duration)
        logger.info('Average FPS: %d', avg_fps)
        mot.print_timing_info()
```

**The function being used to return the tracking ID of a human body that is clicked on in the frame**

```
def click_event(event, x, y, flags, boxes):
    global tid #integer variable that holds the target ID of the human body that will be tracked by the robot
    global there #boolean variable that specifies if there is a body being tracked by the robot still in the frame
    flag=False
    if event == cv2.EVENT_RBUTTONDOWN: #detecting a right mouse button click to reset and stop tracking anybody
                there=False
                tid=-1
    if event == cv2.EVENT_LBUTTONDOWN: #detecting a left mouse button click to select a human body in the frame to track
        for box in boxes:
            if (x>=box[1] and x<=box[3]) and (y>=box[2] and y<=box[4]):
                flag=True
                tid=box[0]
                print("Target Selected",tid)
                break
            else:
                tid=-1
```

**Calling the main function**

```
if __name__ == '__main__':
    main()
```

## Part II : Arduino UNO R3

Language: C

Version servo: 1.1.8

Version math: 3.0.2

After sending the coordinates of the center of the selected body to the Arduino Uno R3 a code implemented on it well make the calculations to move the servo motors

**Included libraries**

```
#include <Servo.h>
#include <math.h>
```

**Main function: Defining the needed variables**

```
int width = 640, height = 480;  // total resolution of the video
float xpos = 85, ypos = 140;  // initial positions of both Servos
// define the centers for the object and the hight
int x_mid, y_mid, objheight;
double dist,dx,dy;
Servo x, y; //define the servo motors
```

**Main function: Setup the serial communication and the control pins**

```
void setup() {

  Serial.begin(115200); //Serial bandwidth
  x.attach(5); //control pin for pan
  y.attach(6); //control pin rot tilt
  x.write(xpos); //send initial pan position
  y.write(ypos); //send initial tilt position

}
```

**Main function: Read the inputs from the Xavier**

```
//read the sent data from jetson (x and y for the centers and h for the hight)
  while(Serial.available()){Serial.read();}
  if (Serial.available() > 0)
  {
    if (Serial.read() == 'X')
    {
      x_mid = Serial.parseInt();
      if (Serial.read() == 'Y')
        {
        y_mid = Serial.parseInt();
        if (Serial.read() == 'H')
        objheight = Serial.parseInt();
        }
    }
    else
      return;
```

**Main function: calculate the errors and the distance of the body**

```
dx=width/2 - x_mid;
    dy=height/2 - y_mid;
    dist=-1.4419*double(objheight)+834.42;
```

**Main function: control the servo motors**

```
if ((x_mid > ((width / 2) + 30)) || (x_mid < ((width / 2) - 30))) // this is the range where the center(x) of the body can be in
{
  xpos=xpos+4*(dx/450)*(350/abs(dist)); //every loop this equation is done tell the center(x) in range
  x.write(xpos);
}

if ((y_mid > height / 2 + 30) || (y_mid < height / 2 - 30)) // this is the range where the center(y) of the body can be in
{
  ypos=ypos-3*(dy/225)*(350/abs(dist)); //every loop this equation is done tell the center(y) in range
  y.write(ypos);
}
```

**Main function: boundaries for the servo angles**

```
// if the servo degree is outside its range
  if (xpos >= 160)
    xpos = 160;
  else if (xpos <= 10)
    xpos = 10;
  if (ypos >= 170)
    ypos = 170;
  else if (ypos <= 80)
    ypos = 80;
```

## Usage

To run the code on the Jetson Xavier NX and start tracking, you need to allow access to the USB port the Arduino is connected to for the first time only by running this command. (Replace the # with the USB port number)

```
sudo chmod -R 777 /dev/ttyUSB#
```

Then run the script based on the argument parser discussed above in Part I.

```
python3 track.py --input-uri /dev/video0 --mot
```

# References


[1] NVIDIA®, "Jetson Xavier NX Developer Kit," 2019. [Online]. Available: https://developer.nvidia.com/embedded/jetson-xavier-nx-devkit. [Accessed 11 8 2022].

[2] HP, "HP Customer Support," [Online]. Available: https://support.hp.com/ph-en/document/c03162257. [Accessed 9 8 2022].

[3] icecat, "HP HD-3100 webcam," [Online]. Available: https://icecat.biz/en-my/p/hp/bk356aa-abb/webcams-hd-3100-16667981.html. [Accessed 11 8 2022].

[4] GROBOTRONICS, "Laser Transmitter Module 650nm KY-008," [Online]. Available: https://grobotronics.com/laser-transmitter-module-650nm-ky-008.html?sl=en. [Accessed 9 8 2022].

[5] AliExpress, "DIY Metal Robot U Shaped Horn," [Online]. Available: https://he.aliexpress.com/item/4000711553253.html?gatewayAdapt=glo2isr. [Accessed 9 8 2022].

[6] TinySine, "Multi-Purpose Servo Bracket," [Online]. Available: https://www.tinyosshop.com/index.php?route=product/product&product_id=733. [Accessed 9 8 2022].

[7] ebay, "Long U-Shaped Bracket," [Online]. Available: https://www.ebay.com/itm/154726377989. [Accessed 9 8 2022].

[8] Amazon, "UCTRONICS Pre-Assembled 2 DoF Pan Tilt Digital Servo Kit, Full Metal Bracket for Building Robotic Arms," [Online]. Available: https://www.amazon.com/UCTRONICS-Pre-Assembled-Digital-Building-Raspberry/dp/B085HDYTCQ. [Accessed 9 8 2022].

[9] TowerPro, "MG996R," [Online]. Available: https://www.towerpro.com.tw/product/mg996r/. [Accessed 9 8 2022].

[10] jaybdub, "NVIDIA Developer Forums," 3 6 2019. [Online]. Available: https://forums.developer.nvidia.com/t/jetson-nano-wiring-to-pwm-driver-for-servo-control/75767. [Accessed 9 8 2022].

[11] Arduino, "UNO R3," [Online]. Available: https://docs.arduino.cc/hardware/uno-rev3. [Accessed 9 8 2022].

[12] S. Thornton, "Microcontrollertips," 17 11 2020. [Online]. Available: https://www.microcontrollertips.com/microcontroller-power-source-considerations-arduino-faq/. [Accessed 9 8 2022].

[13] Amazon, "Charmast Portable Charger," [Online]. Available: https://www.amazon.com/10400mAh-Portable-Charger-External-Compatible/dp/B07JYYRT7T. [Accessed 8 9 2022].

[14] GeekAlexis, "FastMOT," 22 Januray 2022. [Online]. Available: https://github.com/GeekAlexis/FastMOT. [Accessed 6 September 2022].