HUMANOID ROBOT CONTROL USING HUMAN POSE ESTIMATION

A PREPRINT

Sirusala Niranth Sai
18095074, B-Tech
Department of Electronics Engineering
Indian Institute of Technology (BHU)
Varanasi-221005
sirusalansai.ece18@itbhu.ac.in

Piyush Sharan
18095087, B-Tech
Department of Electronics Engineering
Indian Institute of Technology (BHU)
Varanasi-221005
piyushsharan.ece18@itbhu.ac.in

Nishant Kumar
18095048, B-Tech
Department of Electronics Engineering
Indian Institute of Technology (BHU)
Varanasi-221005
nishantkr.ece18@itbhu.ac.in

Supervisor: **Dr. Satyabrata Jit**
Professor (HAG Scale)
Department of Electronics Engineering
Indian Institute of Technology (BHU)
Varanasi-221005, INDIA

June 19, 2020

ABSTRACT

One of the key challenges in robotics has been making robust controllers that can track highly dynamic and agile locomotion. The traditional control strategies simply require too much manual tuning and expert knowledge to handle. However, we realize it to be an unwanted complication as we have a great machine to learn from, the human body !! In this project, we propose a new method to control a humanoid robot using pose estimation from a single RGB image. Earlier approaches use sophisticated sensor and motion capture systems to do this effectively. Our setup is robust enough to work with a decent laptop's web camera using limited computing power. The significance of our approach lies in the fact that it requires no technical expertise to control a robot and also paves way for really dynamic motions which are only limited by the demonstrator's ability. The motion from the camera, tracked by the model we use, is mapped to a robot's controller with proper dynamic balancing introduced to keep the robot from losing balance in a rugged environment.

***K*eywords** Convolutional Neural Networks · PyBullet

1 Introduction

Manually controlling a humanoid robot is a strenuous task. As there are so many joints, it becomes difficult to control them all at once. Posture recognition systems have been widely used in various fields like gaming consoles, animation, movies, etc but until the recent advent of machine learning based approaches, they were hardly scalable to any real-life scenarios. Robotics is one such field in which security concerns play a crucial role than the performance itself. It's quite reasonable to conclude that machine learning based approaches are quite accurate and we are just using it as a tool for manual control and not for automation. We are using the Computer Vision model, PoseNet to detect human posture from an image. This posture is then used by the robot to adjust its position to match the human posture from the image. The simplistic as it may sound, the major challenge here was to accurately transform the posture from the human to the bot, as they could be topologically different to some extent. This should be accurately accounted as it could greatly alter the postures and cause undesirable effects.

2 Definitions and terminology

Convolutional Neural Network: A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance to various aspects/objects in the image and be able to differentiate one from the other.

Key abbreviations:
RGB - Red Green Blue channels of an image
URDF - Unified Robot Description Format
SDF - Simulation Description Format

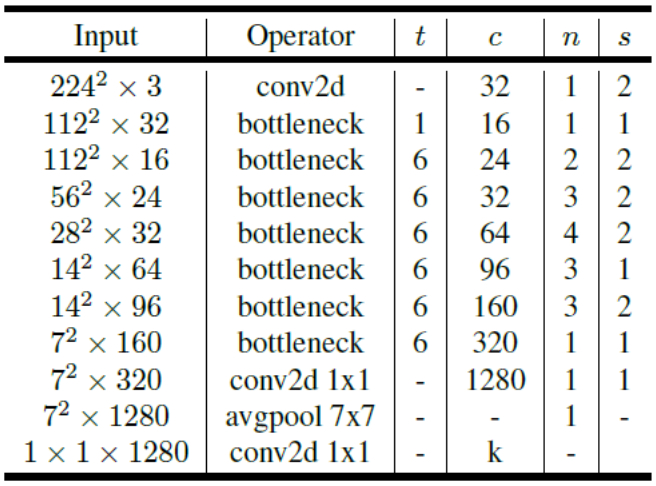
3 Method

Table of contents:
The project can be broadly divided into three parts:
1. Capturing pose from an RGB image using PoseNet.
2. Extracting angles from key points given by PoseNet.
3. Feeding calculated angles into our humanoid robot in PyBullet Physics Engine.

3.1 Capturing pose from an RGB image using PoseNet

PoseNet is a Convolutional Neural Network model that can be used to estimate the pose of a person in an image or video by estimating where key body joints are.
The architecture used is of Google MobileNetV2. MobileNet dramatically reduces the complexity cost and model size of the network, which is suitable for Mobile devices, or any devices with low computational power.
The full architecture of MobileNet is as shown in the Figure 1.

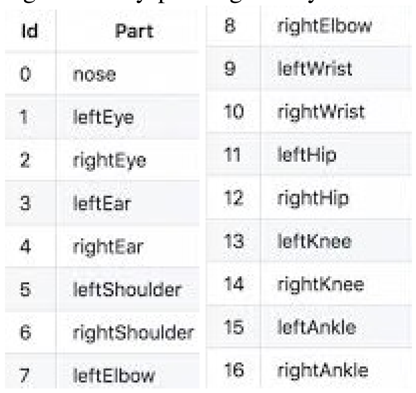
Figure 1: Network architecture of MobileNet
t: expansion factor, c: number of output channels, n: Number of times the layer is repeated, s: stride.



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Pose estimation refers to computer vision techniques that detect human figures in images and videos, so that one could determine, for example, where someone's elbow shows up in an image. The algorithm is simply estimating where key body joints are. PoseNet outputs 17 body parts and the parts are chained in a graph. The key points detected are indexed by “Part ID”, with a confidence score between 0.0 and 1.0, 1.0 being the highest. The keypoints are mapped to their corresponding locations in Figure 2:

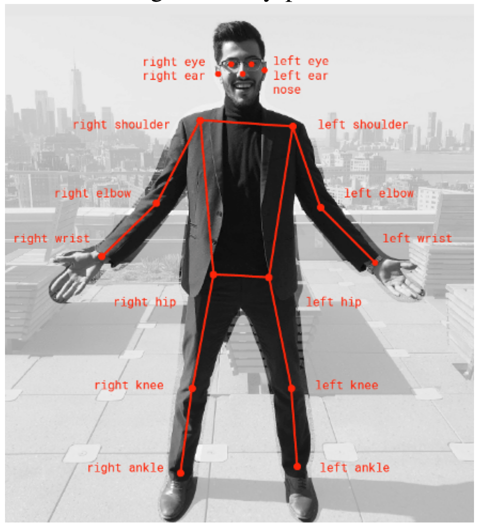
Figure 2: Key-points given by PoseNet



Id	Part
0	nose
1	leftEye
2	rightEye
3	leftEar
4	rightEar
5	leftShoulder
6	rightShoulder
7	leftElbow
8	rightElbow
9	leftWrist
10	rightWrist
11	leftHip
12	rightHip
13	leftKnee
14	rightKnee
15	leftAnkle
16	rightAnkle

At a high-level pose estimation happens in two phases:
1. An input RGB image is fed through a convolutional neural network.
2. An algorithm is used to decode poses, pose confidence scores, keypoint positions, and keypoint confidence scores from the model outputs.
Pose — at the highest level, PoseNet will return a pose object that contains a list of key-points and an instance-level confidence score for each detected person.
Pose confidence score — this determines the overall confidence in the estimation of a pose. It ranges between 0.0 and 1.0. It can be used to hide poses that are not deemed strong enough.
Keypoint — a part of a person’s pose that is estimated, such as the nose, right ear, left knee, right foot, etc. It contains both a position and a keypoint confidence score.
Keypoint Confidence Score — this determines the confidence that an estimated keypoint position is accurate. It ranges between 0.0 and 1.0. It can be used to hide key points that are not deemed strong enough.
Keypoint Position — 2D x and y coordinates in the original input image where a key-point has been detected.

Figure 3: Key-points



Inputs for the single-pose estimation algorithm:

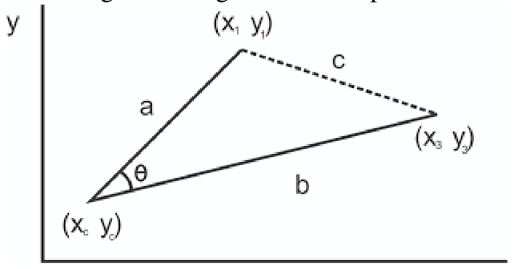
- Input image element — An html element that contains an image to predict poses for, such as a video or image tag. Importantly, the image or video element fed in should be square.

- Image scale factor — A number between 0.2 and 1. Defaults to 0.50. What to scale the image by before feeding it through the network.
- Flip horizontal — Defaults to false. If the poses should be flipped/mirrored horizontally. This should be set to true for videos where the video is by default flipped horizontally (i.e. a webcam), and you want the poses to be returned in the proper orientation.
- Output stride — Must be 32, 16, or 8. Defaults to 16. Internally, this parameter affects the height and width of the layers in the neural network. At a high level, it affects the accuracy and speed of the pose estimation. The lower the value of the output stride the higher the accuracy but slower the speed, the higher the value the faster the speed but lower the accuracy.

3.2 Extracting angles from key-points given by PoseNet

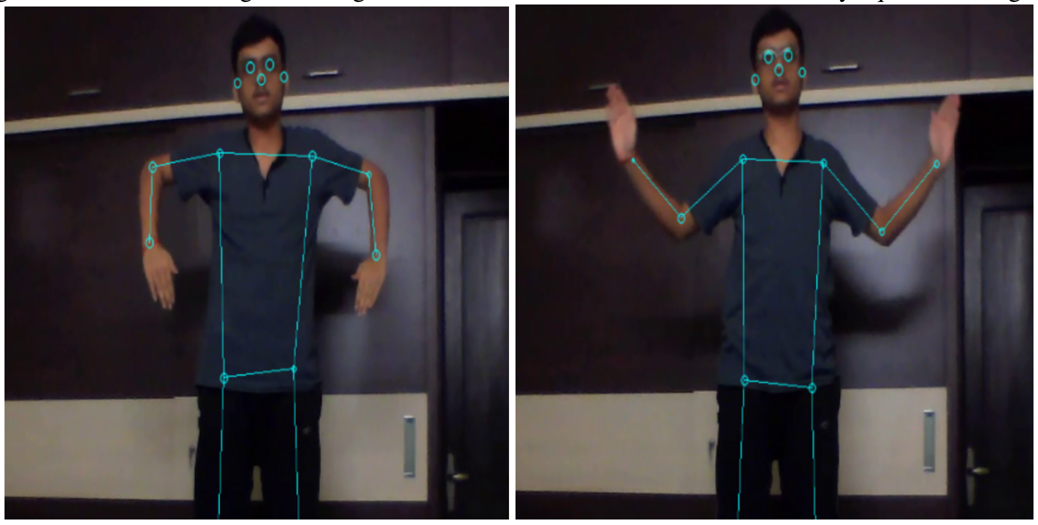
The key points given by our PoseNet are used to calculate different joint angles. In general the angle between three points can be calculated using coordinate geometry by finding slopes of the two lines formed and finding the angles between them.

Figure 4: Angle between 3 points



Directly calculated angles between points would not help here as each joint angle for the robot is to be given with respect to a different frame. So each individual angle was scaled according to the robot frame. There were some cases when the angle was the same for different poses as shown in the Figure 5.

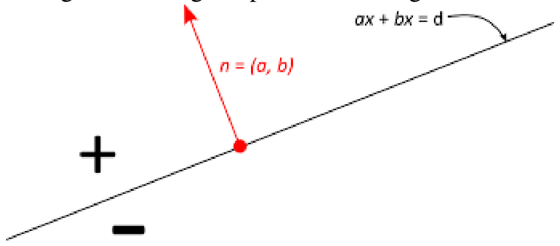
Figure 5: In both the images, the angle made at the elbow is the same and is nearly equal to 90 degrees



We used a different approach to differentiate both the cases in Figure 5. We used the line joining shoulder to elbow joint as a reference line, then the equation of this line is derived using the key-points at shoulder and elbow. By substituting the position of the wrist in the above line equation, we find to which side of the line is our wrist present.

As shown in Figure 6 all the points above the line give a positive value, all the points below the line give a negative value when substituted in the line equation. The two poses in Figure 5 are differentiated using this condition. Here the

Figure 6: Figure showing the positive and negative sides of a line

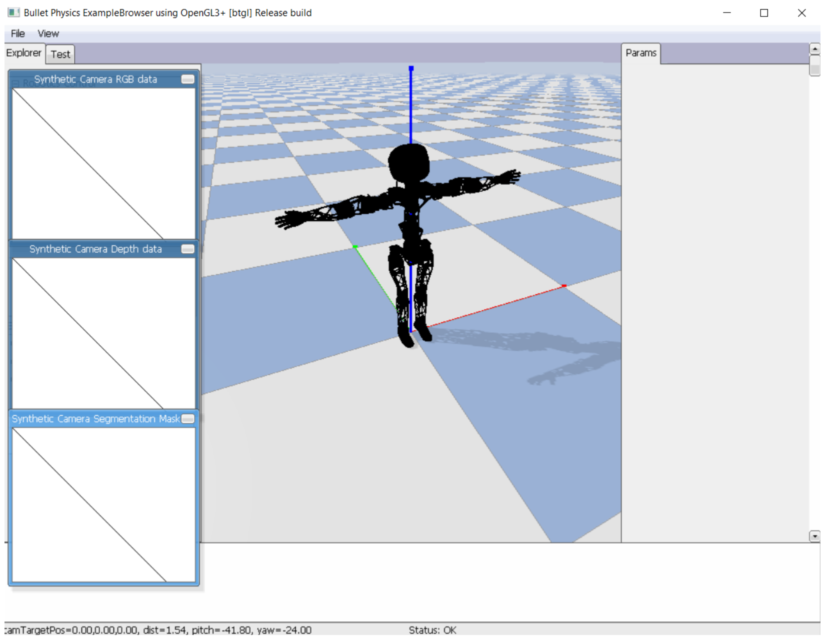


line joining elbow and shoulder is taken as a reference line, the position of the wrist is substituted in this line equation to know which side of the line it belongs to.

3.3 Feeding calculated angles into our humanoid robot in PyBullet Physics Engine:

PyBullet is a Python module for robotics simulation and machine learning, with a focus on simulation-to-real transfer. With PyBullet you can load articulated bodies from URDF(Unified Robot Description Format), SDF(Simulation Description Format), MJCF(Multi-Joint dynamics with Contact XML file) and other file formats. PyBullet provides forward dynamics simulation, inverse dynamics computation, forward and inverse kinematics, collision detection, etc. A simulation in the Bullet physics engine can be easily brought into the real world with some minor tweaks.
As doing it on hardware was not possible due to lockdown, so we used PyBullet as an alternative.
URDF file of Poppy humanoid robot was used in the simulator as it is free, easy to use, and can be easily brought into real-world by 3D printing the STL files of robot parts.

Figure 7: Simulation showing Poppy Humanoid in PyBullet

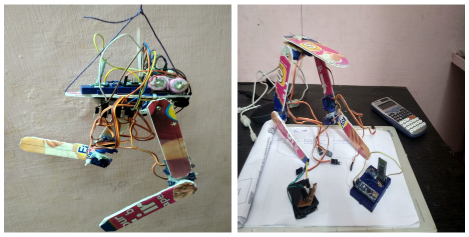

Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build
File
View
Explorer
Test
Params
Synthetic Camera RGB data
Synthetic Camera Depth data
Synthetic Camera Segmentation Mask
camTargetPos=0.00,0.00,0.00, dist=1.54, pitch=-41.80, yaw=-24.00
Status: OK

The angles were scaled according to the humanoid and are used in the ‘setJointMotorControl2’ function of PyBullet to control the joints of the robot by setting desired control mode and position/force/torque.

4 Hardware Validation

Our primary goal was to deploy our strategy in a real-world robot to validate its performance and efficiency. Unfortunately, we were unable to continue our work with the hardware owing to the pandemic situation. However, we would like to share out preliminary prototyping results in hardware development.

4.1 Version 1, Motion capture model

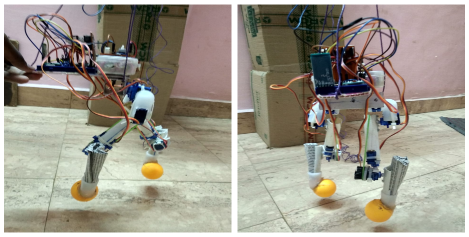


The first prototype was to verify the feasibility of the approach in real-life hardware. As you can see from the picture above, the robot was not build to sustain any form of load but rather was just a proof of concept of our Pose based control architecture. However, owing to a practical robot design we equipped the robot with an Inertial Measurement Unit and Bluetooth functionality for serial communication. The detailed specification of our electronics systems is as follows.

Table 1: Electronic System specification

S.No	Component	Hardware Used
1.	Micro controller	Arduino Nano,with Atmega328p micro controller
2.	Inertial Measurement Unit	MPU 6050,with 3 axis accelerometer and gyroscope
3.	Bluetooth	HC -05 Module for serial communication through UART

4.2 Version 2, Rigid Body Model



In the second installment of our robot, we enhanced the mechanical design of our robot to achieve a sense of static stability. This prototype aimed to make the robot walk with tethers and support structures. Unlike, version 1 our second iteration on the problem proved to be more inclined towards addressing the underlying dynamics of our model. We were effectively able to trace a full stretch gait by pose based demonstration. Unfortunately, we were not able to continue our work in hardware, as we left it in our hostel rooms.

5 Applications

The controller we have designed could be easily extended to a variety of robots.The robots can displace humans in terms of capability there is still a great unfilled void in reliable automation of such systems . Hence, the need for a efficient human control is inevitable and a suitable example would be in the Military – Unmanned Army, bots, etc.
These control systems could greatly enhance the capabilities and deployment of robots in Hazard prone areas and disaster recovery tasks.
It will be difficult for autonomous robots to perform some tasks in an unpredictable environment, rather it would be easy to control the robot using our body pose as a remote.

6 Future work

Using a 3D PoseNet: the robot will be able to move in 3 dimensional space using the 3D pose predicted by our 3D model. But, due to computational limitations, we were not able to use 3D PoseNet for the project. For the case of a 3D PoseNet, keeping a separate PID(Proportional Integral Derivative) based controller to balance the robot in all directions would make it more robust.

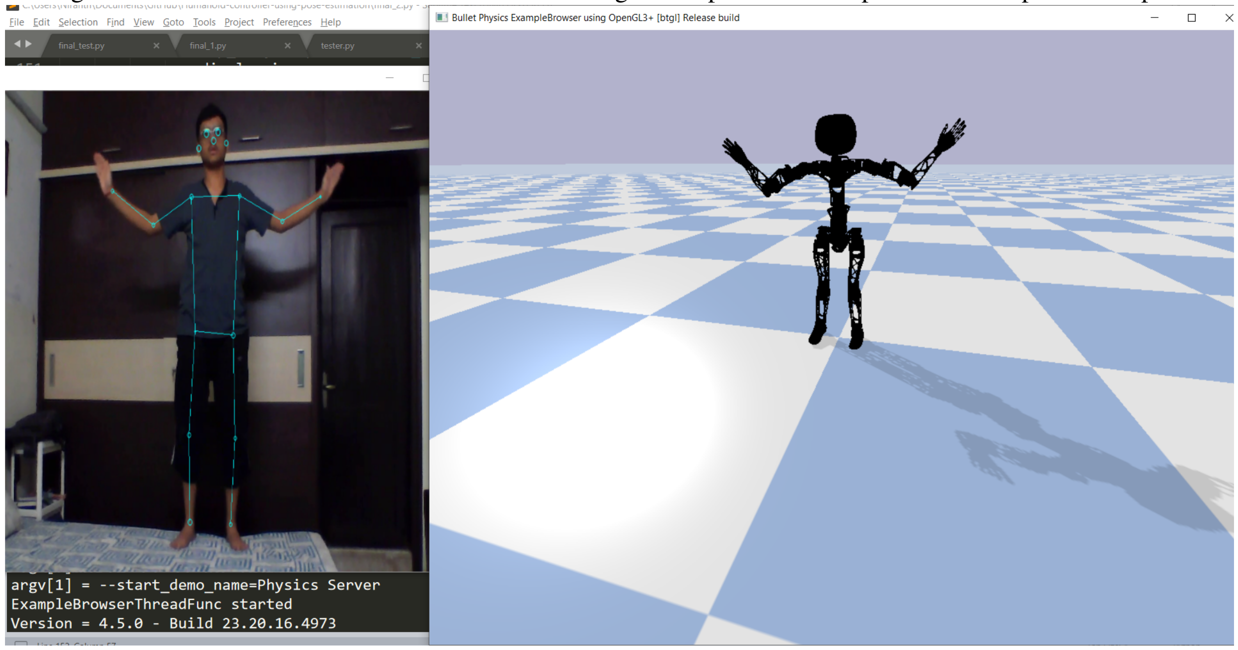
7 Results and Conclusion

In this work we take a novel approach, to address the problem of robot controller design. We show the robustness and scalability of our idea by validating our results in the Pybullet physics simulation engine. We are confident that it could be readily transferred to a real-life robot as we experimented with the 3d model of a prefabricated real-world Humanoid robot named Poppy. Though the model was tested on a single robot it is nowhere limited in its capabilities and could easily generalize a variety of humanoid robots irrespective of the design, topology, and other mechanical parameters of the robot. The idea is not limited to 2D control and could be easily extended towards a 3D controller by adding a bi-camera structure placed perpendicular to each other. However this creates a additional cost overhead, instead we could use an alternate method by using depth map estimation from a 2D RGB image from a monocular camera using General Adversarial Network-based approaches.

We have kept all the files made of this project in the below repository
`https://github.com/nishantkr18/Humanoid-controller-using-pose-estimation`

Demo video: `https://drive.google.com/file/d/1PMeE-UpXgCeP0Dca3F1I_jvnYpxc2cBa/view`

Figure 8: A screenshot of the final simulation showing the output humanoid pose and the input human pose


File Edit Selection Find View Goto Tools Project Preferences Help
final_test.py
final_1.py
tester.py
Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build

```
argv[1] = --start_demo_name=Physics Server
ExampleBrowserThreadFunc started
Version = 4.5.0 - Build 23.20.16.4973
```

References

[1] George Papandreou, Tyler Zhu, Liang-Chieh Chen, Spyros Gidaris, Jonathan Tompson, Kevin Murphy PersonLab: Person Pose Estimation and Instance Segmentation with a Bottom-Up, Part-Based, Geometric Embedding Model

[2] Zhe Cao, Tomas Simon, Shih-En Wei and Yaser Sheikh
Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields

[3] E Coumans, Y Bai
pybullet, a Python module for physics simulation in robotics, games and machine learning

[4] https://github.com/rwightman/posenet-python

[5] https://github.com/tensorflow/tfjs-models/tree/master/posenet

[6] https://github.com/poppy-project/poppy-humanoid

[7] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., et al.:
TensorFlow: Large-scale machine learning on heterogeneous systems (2015) Software available from tensorflow.org