Perception: Human detector and tracker

Anubhav Paras
M. Eng. Robotics
University of Maryland
College Park, MD, 20742
Email: anubhavp@umd.edu

Sakshi Kakde
M. Eng. Robotics
University of Maryland
College Park, MD, 20742
Email: sakshi@umd.edu

Siddharth Telang
M. Eng. Robotics
University of Maryland
College Park, MD, 20742
Email: stelang@umd.edu

I. Introduction

A. Overview

Object detection has become an essential part of all robotic applications. A robot perceives the environment through its sensors and on the basis of the data collected, it performs some action. Specifically, human detection is a core problem in object detection. Avoiding collision with humans during the work process is a critical task for any developer who is designing the robotic application. This avoidance is strictly enforced to avoid any harm, damage or loss of (in the worst case scenario) human life. To add to this, object detection and particularly human detection in a dynamic environment is a very challenging task and demands high precision.

Numerous human object detectors have been developed till date which work very smoothly and accurately and their applications are growing rapidly. In addition it is one of the most popular research topics. With the advent of machine learning it has become easier to design these trackers given a large data set. However, it takes a long time to train such models and require high computational power and efficiency.

ACME Robotics would be launching their robotics based product next year and a human detection and tracking system is the heart of that product. They have given us the complete ownership of designing and developing this detector and tracker. For this application, we aim to use Histogram of Oriented Gradient (HOG) features with a Support Vector Machine (SVM) classifier.

B. Deliverable

- A module in C++ to detect and track humans using feed from a monocular camera.
- Overview of proposed work including timeline, risks, mitigations and UML diagrams.
- GitHub repository with README.
- Continuous integration and code coverage with TravisCI and Coveralls.
- Memory leaks check and profiling using Valgrind.
- Developer-level documentation.

C. Evolution of Software Project Management Plan (SPMP)

The project is going to be rolled out in one release and will comprise of a single version (HDTv1.0).

D. References

- N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 2005, pp. 886-893 vol. 1, doi: 10.1109CVPR.2005.177.
- M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt and B. Scholkopf, "Support vector machines," in IEEE Intelligent Systems and their Applications, vol. 13, no. 4, pp. 18-28, July-Aug. 1998, doi: 10.11095254.708428.
- J. C. Nascimento, A. J. Abrantes and J. S. Marques, "An algorithm for centroid-based tracking of moving objects," 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258), 1999, pp. 3305-3308 vol.6, doi: 10.1109ICASSP.1999.757548.

E. Definitions and acronyms

- HoG : Histogram of Oriented Gradients
- SVM : Support Vector Machine

F. Methodology

Initially, we aim to train our model to extract features which would recognize humans in any given data set. We plan to use the INRIA Person dataset to train our model. Given the data, a HoG feature descriptors would be generated. Next, these HoG vectors or features will be used to train a SVM classifier to detect the location of a human in any image. An error function will calculate the error made in classification which would be then fed to the classifier as a feedback to improve the accuracy. After the training is completed, the classifier would be able to detect the human position and draw a bounding box around it in real (or near-real) time. In addition to this, the output of the classifier would be given to a tracking system to keep track of human(s) in real-time. Moreover, the tracked output will also be converted from image and camera frame to robot's frame for further processing. The output of the tracker will be in image frame. Using the camera's intrinsic matrix, the result can be converted to the camera frame. The transformation between camera and robot is static. This transformation can be used to get the result in robot's frame. Please note that using the inverse of camera matrix, we get the coordinates in camera frame up to a scale. Refer fig.1. (Figure obtained from here).

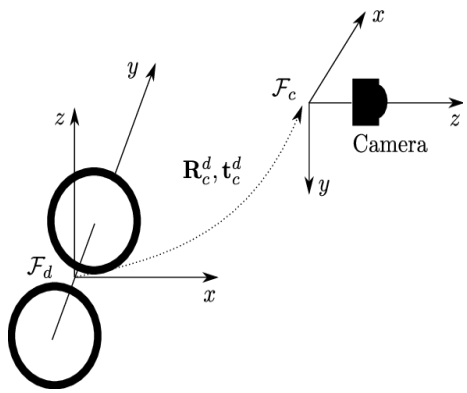

Fig. 1. Camera Robot frame

II. Project organization

A. Process model

Refer fig. 2 for the class UML diagram of the proposed method.

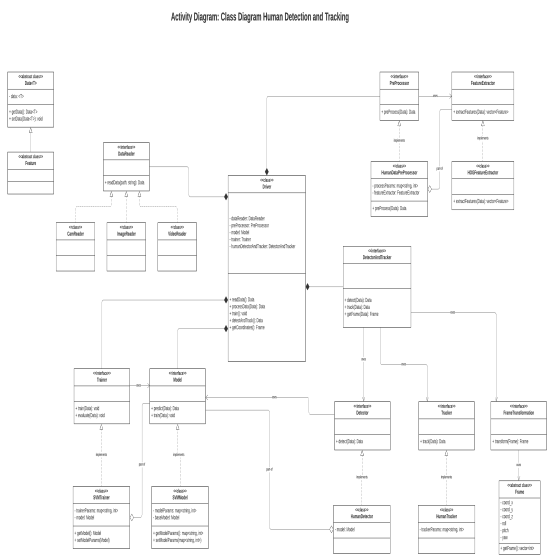

Fig. 2. Class Diagram

B. Organizational structure

- Anubhav Paras (Navigator)
- Sakshi Kakde (Driver)
- Siddharth Telang (Design Keeper)

C. Organizational boundaries and interfaces

- Since the proposed solution depends on the gradient of the image, the algorithm can be sensitive to lightning conditions.
- Might not work in night or low light conditions.
- HoG is not scale invariant.
- Missed detections: It may happen that the model sometimes misses the detection of human presence. In such a case, deploying more than one and different models can be more useful.
- False and duplicate detection: Can be reduced by training the model such that the parameters are well tuned and the loss on validation data set is minimum.
- Unreliable detection boundary: Can be made accurate by selecting an optimum classifier, like the Bayes classifier.
- Flickers in detection: Filter can be deployed for a better estimate.

III. Managerial process

A. Assumptions, dependencies and constraints

- Lighting condition is good.
- The images gradients are well defined.

B. Risk management

- In case the tracker does not work as expected, we intend to simply display the bounding boxes.
- In case the human detection fails, we will detect just the human faces.

C. Monitoring and controlling mechanisms

Git will be used to maintain all versions of our software changes. Build for all the changes will be verified using TravisCI and Coveralls to make sure all our changes are building successfully. To keep a track of correct incremental changes, designed unit tests will be used as initial passing criteria.

IV. Technical process

A. Tools and techniques

- Ubuntu 18.04(LTS)
- C++ 14+
- CMake
- Git
- Travis CI
- Coveralls

B. Software documentation

Doxygen will be used to maintain a developer level documentation.

V. Work packages, schedule and budget

A. Work packages

- OpenCV 4.5.0
- Eigen 3.4.0
- Boost 1.65
- GTest
- GMock

B. Resources requirements

The code will be tested on Intel Core i7-10750H CPU (2.60GHz × 12) equipped with GeForce RTX 2060 GPU and 16 GB.

VI. Appendix

Group - 1 [Anubhav Paras, Sakshi Kakde, Siddharth Telang]					
Human Obstacle Detector and Tracker					
Product Backlog					
Unique ID	**Task**	**Sprint**	**Estimated time (minutes)**	**Time after Iteration 1**	**Time after Iteration 2**
1	**Plan and Design**	**1**	**1976**	**0**	**0**
1.1	Literature Survey about about human detection and tracker	1	120		
1.2	Arrange datasets for detection model	1	60		
1.3	Discussing and defining overall detailed deliverables	1	60	0	0
1.4	Discussion regarding the end-to-end flow of the project	1		0	0
1.4.1	Defining assumptions, constraints and back-up plan for the project	1	240	0	0
1.5	Setting up the git repository and CI badges (Travis and Coveralls)	1		0	0
1.5.1	Configuations for Travis and Coverall in travis.yml	1	15		
1.5.2	Add Licenses	1	15		
1.6	Pushing the initial template for the code and configuration files	1	15	0	0
1.7	Forking the repository by the other team members	1	1	0	0
1.8	Initial design diagrams:	1		0	0
1.8.1	Designing activity diagrams for the process model	1	240		
1.8.2	Designing UML class diagram based on activity diagram	1	360		
1.9	Preparing the design and software documentation for the project	1	250		
1.10	Update the README with overview and documentation	1	100		
1.11	Add dependencies in project CMakeList.txt	1	10	0	0
1.12	Code template for class design as per the UML diagrams	1	120	0	0
1.13	Unit tests for the class stubs created	1	240	0	0
1.14	Static code analysis using cppcheck and cpplint	1	30		
1.15	Discussion for confirming the design and future plan for implementation	1	100	0	0
2	**Implementation**		**1830**	**0**	**0**
2.1	Review the design documentation and UML diagrams	2		0	0
2.1.1	Revise and update the initial UML diagrams	2	120	0	0
2.2	Revise the unit tests as per the revised design	2	180	0	0
2.3	Implementation of the classes as per the the activity diagrams	2		0	0
2.3.1	Implementing the DataReader class	2	120	0	0
2.3.2	Implementing the Model and Trainer class	2	120		
2.3.3	Implementing the Detector class	2	120		
2.3.4	Implementing the Tracker class	2	120		
2.3.5	Implementing the FrameTransformation class	2	120		
2.3.6	Implementing the Driver class	2	120		
2.4	Training the model to detect humans using traning data	2	240	0	0
2.5	Evaluating the human detection model using test data	2	60		
2.6	Testing the overall detection and tracking pipleline	2	120		
2.7	Check if the implementation meets the requirement	2			
2.7.1	Planning and implementing the back-up plan if testing produces unexpected results	2	240	0	0
2.8	Generating doxygen documentation for the code implementation	2	30	0	0
2.9	Static code analysis using cppcheck and cppling	2	30	0	0
2.10	Memory management analysis and profiling using valgrind	2	30		
2.11	Pushing the code to git	2	30	0	0
2.11.1	Verifying the builds and coverage using Travis and Coveralls	2			
2.12	Updating the documentation	2	30	0	0
	Remaining effort			**0**	**0**
	Total effort time				**0**

Use index colors to show revisions on task time	
Index	**Information**
	Target Time
	Revised Target Time
	Actual Time Taken
	New Task Added