AUTONOMOUS ROBOT CONTROL WITH DSP AND VIDEO CAMERA USING MATLAB STATE-FLOW CHART

Aviv Lichtigstein, Roy Or-El*, and Arie Nakhmani*

Control and Robotics Lab, Department of Electrical Engineering, Technion – Israel Institute of Technology

* First 2 authors made an equal contribution to the paper

ABSTRACT

This paper presents an embedded system for autonomous control of iRobot Create that consists of real time target tracking and robot navigation. Tracking and navigation algorithms were implemented using the TI DM6437 EVM board. The robot's performance demonstrates that a visual tracking and navigation can be efficiently and robustly implemented, and TI DSP is suitable for running both algorithms simultaneously due to its low power consumption and high-speed performance. The proposed tools have a steep learning curve, and they can be used as a platform for the further development.

1. INTRODUCTION

Autonomous systems have been a very active research area over the past decade. One of the main reasons for the massive research activity in this area is its important applications, especially in security and safety devices. For many of these applications a low-cost, accurate and robust tracking and navigation solution is required.

Though a well studied area, tracking and navigation based on a video source still holds many scientific and technological challenges. Maintaining a fast and smooth navigation of the autonomous system is complicated since we need to handle a dynamic environment. Target tracking can be time consuming and, as a result, the delayed robot reaction may produce errors. For embedded applications, avoiding excessive calculations and large memory consumption is another important challenge.

Previous works of robot navigation with a video camera were implemented using two processing units. Xiao *et al.* [1] suggested an omnidirectional vision system to facilitate vision-based multiple robot coordination, which uses TI F2812 DSP for implementing the robot control system, and a host PC for image processing. Sawasaki *et al.* [2] proposed a stereo-vision-based system that performs feature extraction and block matching operations for autonomous navigation. It has an image processing dedicated board with TI C6713 DSP and an ALTERA Stratix FPGA. Masar and Gerke work [3] presents a neuro-fuzzy controller for trajectory following. Although there is a camera in their robot, a vision algorithm was not presented.

Autonomous navigation systems are not a new area in our lab. Several researches, which involved C2000 series DSPs, were implemented in the past. Those works included only distance sensors and as a result the performances were less accurate.

The algorithms of visual tracking using DSP were proposed earlier, *e.g.*, by Roichman *et al.* [4]. This algorithm for real time pedestrian detection and tracking is uniquely combines background subtraction and temporal differencing for motion detection, and tracks targets by predicting their location according to their motion vector. The algorithm for detection and tracking [4] have been implemented on DM643x family of DSPs. Unfortunately, this algorithm assumes a static camera, which is not suitable for our robot.

Our prototype system uses a fixed camera and background geometrical properties for robot navigation. TI DM 6437 is responsible for target detection, target tracking, and robot navigation. The chosen solution comprises the advantages of using a single code development environment of MATLAB Simulink. This allows achieving system's modularity, expandability, and a user friendly interface. The modularity and expandability enables future implementation of more complex detection and tracking algorithms.

The classical open loop approach of "detect and drive", where the target is detected in the video frame, the robot's trajectory is computed, and the commands are sent to the robot, may be inefficient and not accurate. In the proposed algorithm, the DSP constantly receives feedback from the camera, thus the obtained trajectory is smooth, and the robot is able to robustly navigate after a moving target. Also, this allows accuracy in tracking and navigation.

The prototype described in this paper defines the target to be a red, blob-like, target. By using the proposed algorithms, the robot detects, tracks, and navigates until it reaches the target, and stops.

Section 2 outlines the proposed target detection, tracking and navigating algorithm. Sections 3, 4, and 5 describe the implementation, results, and conclusion respectively.

2. ALGORITHM

Our embedded system is based on target detection, tracking, and navigation of the robot to the target. First, we present the target detection and tracking algorithms. Afterwards, we present the proposed navigation method.

2.1 Target detection and tracking

Target detection is obtained by analyzing the video frames captured by the camera. Camera works in an interlace mode with the resolution of 720×576 pixels. Therefore, data arrive in a YCbCr 4:2:2 format [5]. Data is deinterlaced and should be transformed from YCbCr to RGB color space, but because of uneven matrix sizes, it cannot be transformed directly. Linear extrapolation of the Cb and Cr channels, thus, reconstructing the image back to 4:4:4 format turned out to be too slow and damaged real time execution of the algorithm. Instead, the Y channel was decimated by keeping only even rows of the image. The computed target's center of mass (*centroid*) coordinates, were adjusted accordingly.

A binary mask is than created for each frame in order to find all the red targets in the frame. Any desired color can be masked by a similar procedure. Red masking is performed by setting two thresholds T_1 and T_2 ($T_1<T_2$), using R, G, and B channels:

$$\text{red mask} = \begin{cases} 1, \text{ if } R > T_2, G < T_1, B < T_1 \\ 0, \text{ otherwise} \end{cases} \quad (1)$$

After the red mask has been formed, it is filtered to eliminate any anomalies that may have been caused by lightning conditions, clutter, and noise. Median filter has been chosen due to its good performance in salt and pepper noise filtering, which suits to our purpose.

After the filtering, blob analysis is performed to find the centroid of each red blob. Centroids of blobs smaller than 50 pixels are ignored.

We assume that lightning conditions can cause some of the pixels to appear in a slightly different color than their original color. Therefore, large target can break into fractions during the masking process. Morphological operations such as *imclose* can fix this problem but their implementation using Simulink has the drawback of not being very computationally efficient, and cannot be part of a real time system.

In order to address the problem above, we propose to unify the centroids of any broken targets. Target's true centroid is approximately estimated by the location of its fractions centroids. Centroids are unified by creating a smart labeling algorithm that detects neighboring groups of centroids and gives them the same label. The algorithm scans all the centroids found. If a number of centroids are within a relatively small predefined area, the algorithm assigns the same label to the relevant centroids. Two centroids that are relatively far from each other get different labels. Afterwards, the average centroid is calculated for each group of centroids that carries the same label. By that, we have achieved a robust, quick and efficient method that can substitute the *imclose* operation. The purposed solution complexity is $O(N^2)$ where N represents the number of centroids found in each frame. Therefore, many calculations are saved because N is much smaller than the frame size.

To keep tracking the correct target, we have assumed that there is only one target in the first video frame. No such assumptions were made for any other frame. In each frame the target's centroid location is saved. In the next frame a new set of centroids arrives. The centroid most likely to belong to the target is the nearest neighbor to the target's centroid from the previous frame. In that way, the robot is able to ignore multiple targets of the same color, and reach the desired target without being distracted by other similar targets. Therefore, the closest centroid to the previous frame centroid is chosen as the current frame target's centroid. The chosen centroid is stored in memory for the next frame and also used as an input for the navigation.

2.2 Robot navigation

To successfully navigate the robot to the target, we have used the target's centroid found in the detection and tracking algorithm. Our navigation goal is to bring the target's centroid to a desired location in the image. The desired location is a 180x192 rectangle in the bottom middle (BM) of the screen (see Figure 1). This location is set by considering the camera location on the robot and it represents the area just in front of the robot. By the current centroid location we are able to give the robot navigation directions (such as go forward, stop, turn left, or turn right) for each of the predefined Top, Bottom, Left, Middle, and Right regions (TL, TM, TR, BL, BM, BR).

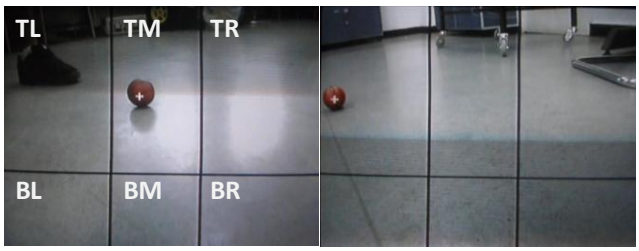

Figure 1: Robot's point of view. Left - *go forward* area; Right - *turn left* and *go forward* area.

The robot turning radius is also set by the centroid location on the screen. Note that when the turning radius is set to R, the robot will move in circles of radius R. for straight line motion, one should set $R=max\ (=2^{15}mm)$. The turning radius was calculated using two heuristic geometrical considerations. First, the radius should decrease when the centroid is going away from the center column of the screen. Second, the radius should increase when the centroid is going towards the upper part of the screen, and that is because targets in the bottom part of the screen are closer to the robot than targets in the upper part of the screen. Turning radius is calculated by:

$$R = \begin{cases} 2^{15}, \text{ if } x_C \in TM \text{ or } x_C \in BM \\ 2000 - \frac{1800 y_C}{576} + \left(\frac{5 y_C}{576} - 5.55\right)(360 - x_C), \text{otherwise} \end{cases} \quad (2)$$

Where (x_c,y_c) is the centroid column and row coordinates, and negative radius represents a right turn. Note that (1,1) is a top left corner pixel.

If no target was detected, the robot turns around his axis to acquire a new target. Once a target is found the robot automatically starts navigating towards the target using the method above.

However, if the robot loses its target, there is a small waiting period before the robot starts searching for a new target. By that we achieve more robustness in cases where the target is hidden from the camera for a small amount of time. The robot stands still and waits for the target to reappear for the waiting period, which is user defined, before it starts looking for another target.

3. IMPLEMENTATION

The prototype system was implemented using the TMS320DM6437 EVM [6] board, the iRobot Create [7], and a SENTech STC-635TC video camera. The EVM board includes a DM6437 DSP with a UART serial port, composite video inputs and outputs, and many other peripherals.

The EVM board is mounted on a plastic base above the robot's cargo bay, using spacers to support it. A steel framework was built, to attach the video camera to the system, and avoid camera vibrations, that can be caused by the robot's movement. The camera sits on an aluminum pole which is perpendicularly connected to the center of the steel framework. A bolt secures the camera into the pole and keeps the camera at a fixed position (see Figure 2).

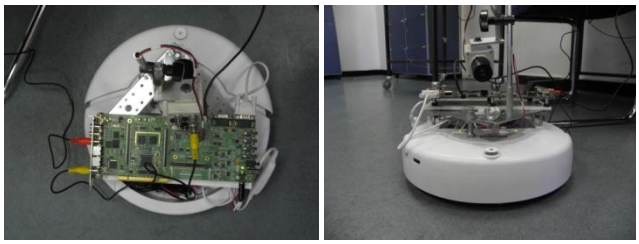

Figure 2: Left – system top view, Right – system front view.

The robot and DSP are communicating via RS-232 protocol. Connection is done by a special cable which is supplied with the robot and connects to the UART connector at the EVM side and to the Mini-DIN connector at the robot's side. The video camera is connected to the EVM board composite video input.

The robot's instructions are composed of one byte Opcode and a number of data bytes that varies for each instruction. Robot's movement can be controlled by the *Drive* instruction which has 4 data bytes, 2 for velocity and 2 for turning radius. For additional details see [7].

Tracking and navigation algorithms were designed in MATLAB Simulink and MATLAB State-Flow chart environment. The described system uses the Simulink C-code generator toolbox to generate an automated high level real time environment for the DM6437 processor.

3.1 Detection and tracking

Implementation of target detection and tracking is done by MATLAB Simulink blocks, including blocks from Target support package specially designated for DM6437 EVM board. In addition, Embedded MATLAB code has been used. The operation of YCbCr to RGB conversion is implemented using MATLAB embedded function.

Tracking is done by comparing the distances between each centroid found in the current frame and the chosen centroid from the previous frame. The centroid with the shortest distance to the previous frame centroid is chosen as the current frame centroid. The chosen centroid is transferred to the navigation part of the algorithm, and also saved in memory to help determining the next frame centroid.

3.2 Robot navigation

The frame was divided into 6 areas: turn left (BL), turn left and go forward (TL), turn right (BR), turn right and go forward (TR), go forward (TM), and stop (BM) (see Figure 1). The turning radius is determined and transferred to the robot as described in Section 2. For debugging, an external monitor was connected to the composite video output of the EVM (see Figure 2).

Rate Transition block is used to synchronize between tracking algorithm sample rate, and navigation algorithm sample rate. Two unsynchronized sample rates can cause timing problems which in turn cause undetermined data transfer. The block ensures that data transfer between the two algorithms will remain determined while the system is running (see Figure 3).

Truth Table block is used to define the conditions regarding the location of the centroid in each frame. The output of the block specifies in which area of the frame the centroid is located by the following rules:

if $x_c = y_c = 0$ then *Scan* for new target

if $x_c \in TL$ or $x_c \in BL$ then *Turn Left*

if $x_c \in TR$ or $x_c \in BR$ then *Turn Right*

if $(x_c, y_c) \in TM$ then *Drive Forward*

if $(x_c, y_c) \in BM$ then *Stop*

The State Flow Chart block receives the radius variables, as they were calculated according to Section 2, and the output of the Truth Table (see Figure 3). Note that the radius is represented by two bytes of data.

The State Flow Chart is the mind of the system. In each frame, the chart decides how the robot should act. Each of the possible inputs is translated to a binary condition in a state flow chart which leads to different state in the chart. In each state, a different command vector of five bytes is sent to the robot through the UART.

The state machine structure consists of 26 states, and it is built to operate autonomously, therefore, all the optional dynamic scenes were considered. Furthermore, the state flow chart supports an advanced set of commands which

are sent to the robot. Scanning mode and internal timer are implemented by creating combination of new states, and they enable the system to operate independently.

The location of the centroid is changing in each frame, and as a result the state in the State Flow Chart is updated, and a new command is sent to the robot. The camera frame rate allows for the update rate of several times per second.

The extensibility and readability of the state flow chart creates a unique simple autonomous control environment which can easily be suited to new complicated tasks.

4. RESULTS

The system with the DM6437 EVM board and standard video camera, both connected to the iRobot was field tested at various real scenarios and different lighting conditions. Targets were detected and tracked accurately, and the robot reached its target every time. The results of successful navigation can be seen in Figure 4.

The robot is able to function for more than two hours with a single battery charging. This demonstrates power efficiency of the proposed robotic framework.

The demonstration of robot abilities can be seen on the internet [8]. The proposed algorithms took less than 4% of DSP memory resources, thus they can be expanded easily to more sophisticated projects.

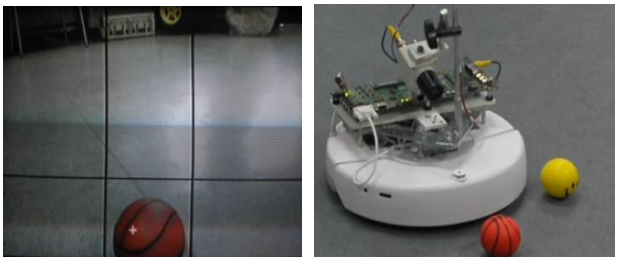

Figure 4: Left – *stop* command, Right – *stop*, external view.

5. CONCLUSIONS

This paper presents a low cost algorithm for target tracking and navigation, and its implementation using DM6437 EVM board, video camera, and iRobot Create. The system operates autonomously, and run in real time.

The advantage of reevaluating the location of the target several times per second, and updating the commands which sent to robot from a clearly defined state machine, supported the accuracy of the real time embedded system.

System's implementation main advantage is in its unified development environment, which enables very large modularity in algorithm's design. Simulink lets the user assemble large and complex systems in very short time and thus, the proposed robot and DSP platform can be used as a base for future development of more complex tracking and navigation algorithms.

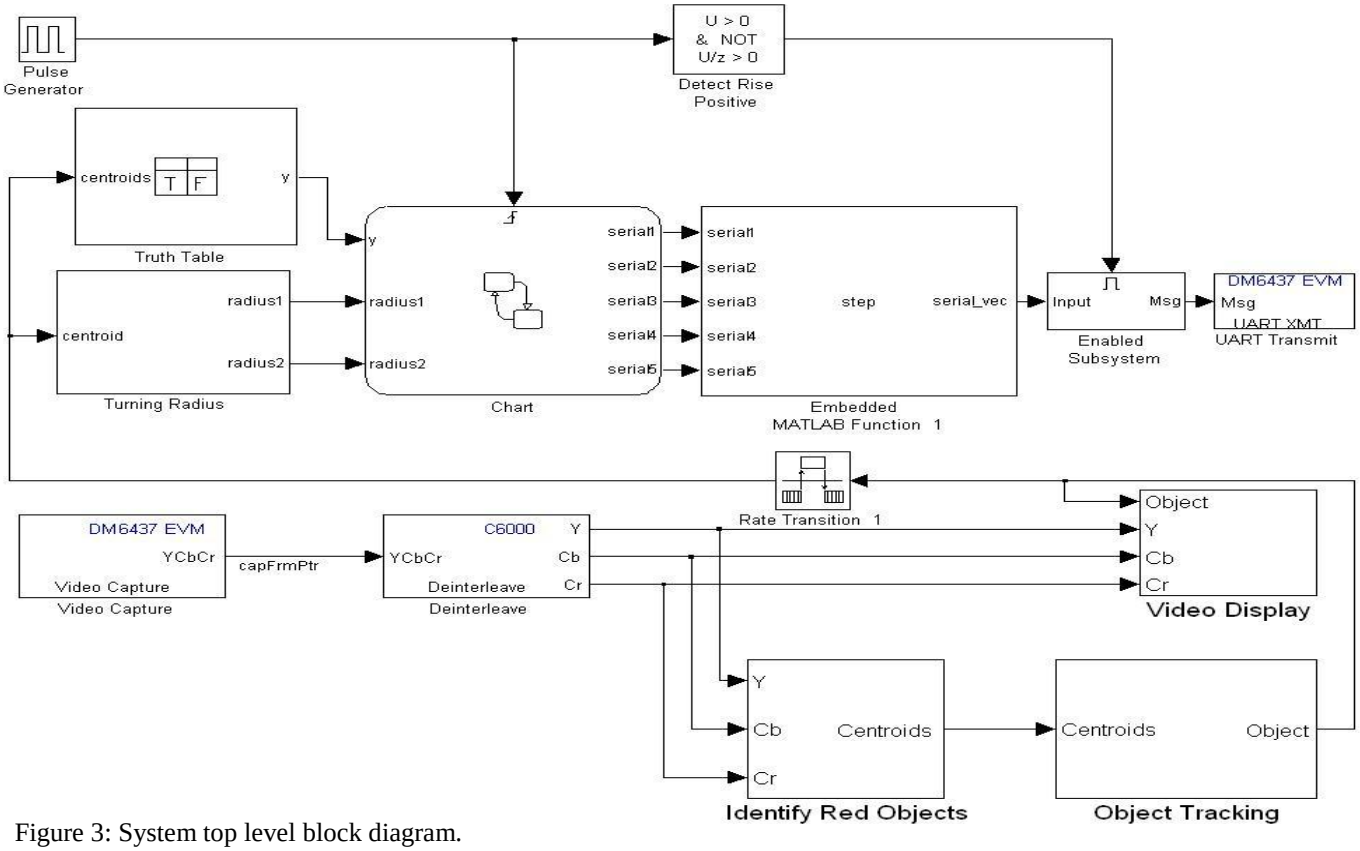

Figure 3: System top level block diagram.

6. ACKNOWLEDGMENT

The authors thank Koby Kohai, Orly Wigderson, Roey Hochman and Eran Domb from Control and Robotics Lab for their help and support to our work.

A special thanks to Yair Moshe and Nimrod Peleg from the Signal and Image Processing lab for contributing their knowledge in DSP and MATLAB Simulink.

REFERENCES

[1] J. Xiao, A. Calle, J. Ye, and Z. Zhu, "A Mobile Robot Platform With DSP-based Controller and Omnidirectional Vision System," IEEE International Conference on Robotics and Biomimetics, 2004.

[2] N. Sawasaki, M. Nakao, Y. Yamamoto and K. Okabayashi, "Embedded Vision System for Mobile Robot Navigation," Proceedings of IEEE International Conference on Robotics and Automation, 2006.

[3] I. Masar, and M. Gerke, "DSP – Based Control of Mobile Robots", Proceedings of European DSP Education and Research Symposium, 2004.

[4] E. Roichman, Y. Salomon, and Y. Moshe, "Real-Time Pedestrian Detection and Tracking," Proceedings of the 3rd European DSP Education and Research Symposium, Tel-Aviv, Israel, 2008, pp.281-288.

[5] K. Jack, "Video Demystified: A handbook for the digital engineer," 4-th edition, Elsevier 2005.

[6] TMS320DM6437 Digital media processor, Texas Instruments, SPRS345D, November 2006, revised June 2008.

[7] iRobot Create, Open interface – V2, iRobot corporation, 2006.

[8] A. Lichtigstein, R. Or-El, (2010), "Autonomous robot control with DSP and video camera," Available: http://www.youtube.com/watch?v=wQ6yQNlbUqo