# TORO: A new approach to implementing a vision-capable robot

Michael Reiser (mreiser@ufl.edu), Dr. A. Antonio Arroyo, Dr. Eric M. Schwartz
Machine Intelligence Lab, University of Florida
2000 Florida Conference on Recent Advances in Robotics
May 4-5, Florida Atlantic University

**Abstract**
TORO was built as my project for the Intelligent Machine Design Lab course, associated with the Machine Intelligence Lab at the University of Florida. The challenge was to quickly construct a sophisticated platform capable of advanced vision-driven behavior using low cost components. This paper details the results of this project. It includes descriptions of both the hardware and the software systems and the implementation details.


Figure 1: TORO

**Introduction**
TORO represents a significant effort to construct a fairly ambitious autonomous mobile platform. The robot has an extensive sensor suite, the most significant of which is the camera, since it is a high bandwidth device. Using camera data in a real-time application is not a trivial issue. Therefore the robot has been designed as a color detector/follower. The implementation of this behavior is not nearly the most complex behavior possible with the platform, but it serves as a realizable goal for a project of this type.

The robot is a two-tiered and two-wheeled rolling platform that houses two separate centers of computation. The lower level functions of sensor data collection and motor control are handled by a Motorola 68HC11 running a very short icc11-compiled program. The higher level functions of camera operation, joystick polling, decision making, and user interfacing are handled by a real-time program running on a Pentium-based laptop. This program controls all behavior arbitration and represents all underlying actions in a graphical manner to the user. A significant accomplishment of this project is the simple interface between the two systems, and the clearly defined tasks required of each system.

**Integrated Systems**
A 68HC11 microcontroller board is used to provide the control logic for the drive system and the polling of the low-level sensors. The PC onboard the robot was initially intended to be a full computer, complete with motherboard and all of the flexibility of a PC's peripheral components. This was to be run off of battery power. Unfortunately due to several compatibility issues, a laptop computer is used in the final result. In the design, the PC is used to control the vision system and the communication system. The communication system is an essential feature of the robot. The PC maintains a constant bi-directional communication with the 68HC11 through a serial connection. The logical separation of functions is then simple. The PC uses the 68HC11's incoming sensor data as well as the data from its own set of sensors to generate decisions (e.g. go left, stop). These decisions are then sent to the 68HC11 via the serial port. All communication consists of ASCII text strings. Figure 2 shows the block diagram for this interaction, detailing the persistent control loop present in the robot.

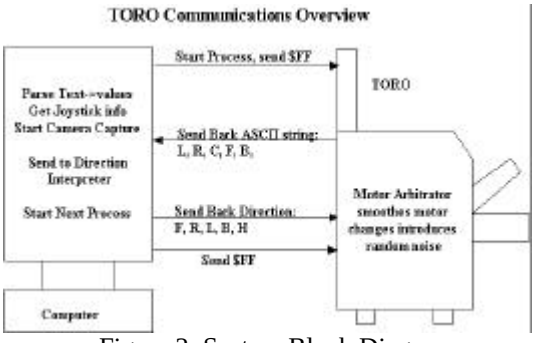

Figure 2: System Block Diagram

**Platform & Actuation**

The platform has been kept very simple. The primary requirement is that there is enough space for all of the hardware and sensors and for possible future expansion. The body is an oval-shaped, tiered rolling platform. The bottom layer contains "storage room" for—the 68HC11, communications board, IR emitters, battery pack, servos, wheels, and bundles of the PC's connection cables. Above this layer is the control layer—this layer contains the PC laptop, all of the sensors or connections to sensors, and all the cables and connections necessary to support the hardware systems.

The actuators used on the robot are fully modified high torque servos, Tower Hobby brand with a listed torque output of 128 ounce-inches at 6V. Since the servos are running at a higher voltage (about 8 volts) I expect that an even greater maximum torque is generated. The servos drive two large (4.5" diameter) model airplane wheels. A centered swivel caster is mounted in the rear for support.

All motor control is performed on the 68HC11. A smoothing function is employed to average the desired and previous values of the motor speeds. The algorithm is very simple. The five directions (right, left, forward, back, and halt) are interpreted as setting up a desired speed for each wheel. The speed is then arrived at by:

```
cur_speedr=(6*cur_speedr +
add_noise(des_speedr))/7;
```

A similar command computes the current speed for the left servo. As can be seen, random noise is added at this point. The noise is on the order of about 20% of the total motor speed value, and adds some needed robustness to the motor control. With the noise, even faulty sensor reading will not cause the robot to trap itself

**Sensors**

TORO contains several types of sensors that allow the machine to intelligently interact with its environment. The most important low-level sensors are necessary to provide for obstacle avoidance. The high-level sensors needed are the robot's "brain," the PC, and the robot's "eyes," the camera.

The robot contains the following sensor types:

- Infrared transmitters/receivers
- Bump Switches
- PC system
- Camera connected to the PC
- Joystick for interacting with the PC

TORO uses three forward-facing IR emitter/receiver pairs. Each emitter is an LED that sends out a modulated 40kHz IR signal which is then detected by the receivers upon reflection. The receivers and emitters are positioned in such a way that there is no direct line of sight between the pair, so all detected IR must be reflected off of an object. The IR receivers used are digital Sharp receivers that have been modified as analog sensors. The output analog voltage is connected to the A/D port of the 68HC11. The analog values, when digitized return a value ranging from 88 to 128. A higher number indicates that the sensor is detecting more IR, and proximity to an object is likely. Figure 3 shows some sample data from the IR detectors when pointing straight at a wall.

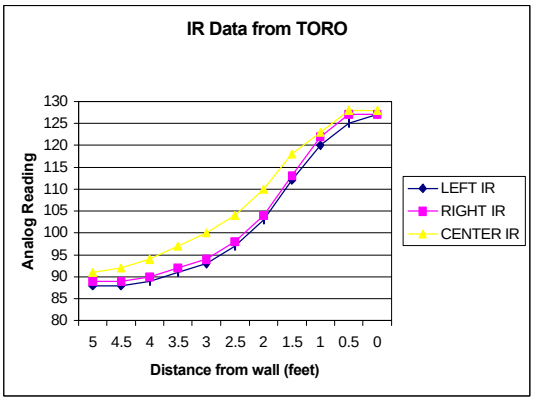


Figure 3: IR Data

It can be seen that the plot of distance vs. IR values is fairly linear over a range of about 4 feet to 0.5 feet away from the wall.

Obstacle avoidance using IR is accomplished by comparing values of IR and moving so as to counteract the highest values. I have found it helpful to set a threshold above which the IR values are "high," in this case a value of 100 seems appropriate since it indicates a distance of between 2 and 3 feet from the obstacle. For example:

- If all three IR > threshold, then reverse
- If left or left/center > threshold, and if left > right, then turn right
- If right or right/center > threshold, and if right > left, then turn left

In the event that the IR data should not be properly handled, or some conditions occur that render this data inaccurate (i.e. dark corners) a system is implemented to back-up the IR. Bump

sensors are used to detect a direct collision with an object and to assist the robot in getting free of that object. The sensors are just switches that are connected via a resistor divider network to the A/D ports of the 68HC11. By properly selecting the values of the resistors the robot can determine the side at which the collision occurred, and move away from that side. A cantilevered wooden bump ring that has a restricted range of movement surrounds the contact switches. The bumpers work by pushing into the switch when contact is made and then using the switch's as a spring to bounce back. This bump system is one of the most successful features of the robot, and is more than capable of distributing the collision to one (or more) of the switches. There are six switches on the robot, three in the front and three in the back, spaced out so that a sensor is at each "corner" and at the direct front and direct rear of the platform. Figure 4 shows an AutoCad sketch of the positioning of the sensors.

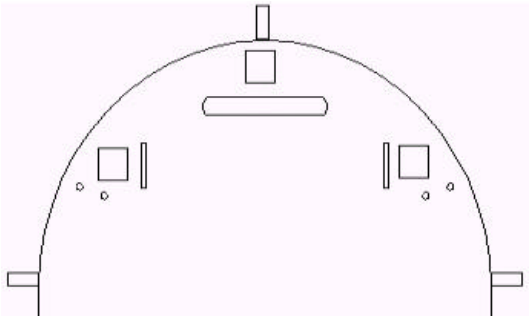

Figure 4: Sensor Positions

In this image the squares represent the IR receivers and the rectangles are the sites for the bump switches.

Here is the data gathered for the values returned by the A/D unit for the bump switches. The front and rear network have been wired in the same way so the values are the same:

| Switch(es) Pressed | A/D Values Range |
|---|---|
| Left | 41-44 |
| Right | 127-129 |
| Center | 76-79 |
| Left, Center | 100-102 |
| Right, Center | 149-153 |

The most complex "sensor" used on the robot is the PC. The PC used is not a 486 desktop system as initially planned—rather it is a laptop. The specifications for the laptop:

- CPU: Pentium 120 MHz
- Memory: 24 MB RAM
- Hard Drive capacity: ~800 MB
- Operating System: Windows 98

In addition all necessary development tools (MS visual Studio 6.0 and ICC11) have been loaded onto the laptop so that the development of the software can occur on the robot itself. The laptop is placed directly on top of the robot as seen in Figure 1. The laptop is connected to the 68HC11 through a serial connection to a communication board that provides for translation from RS232 values to TTL levels.

The camera used is a parallel port connected color Logitech QuickCam 2 (a "web cam") capable of resolutions between 160×´120 pixels to 640×480 pixels. The camera features automatic gain control and is capable of 4-bit to 24-bit colors. The camera performance is about 12fps, with poor detail in low light. The only solution to the lighting problem is to provide a light source on the robot. The camera driver uses the Video For Windows Standard and is thus programmable from within a windows environment. The most successful camera setting are - 160×120, with 256 colors and automatic lighting and hue settings. The camera data is used to perform image processing. A control called "ezVidCap.ocx" is used to handle the acquisition of the images.

The joystick was added to provide for an additional method of controlling the user interface program. The buttons on the joystick have been mapped as a replacement for using the mouse to select certain options in the program. The joystick used is a Microsoft Sidewinder Pro. This joystick is capable of 3D control and has nearly a dozen buttons on it. I implemented the joystick as a 2D, 4 button joystick, and was able to receive x and y coordinates and button presses from the windows API. Since the range of values for each axis is larget—it is a value from 0 to approximately 64k—only the very extremes of the joystick's position for a value are used.

**Behaviors**

The robot has 4 sensor systems, each driving its own behavior. After all sensor data is collected, there is a function that interprets each sensor's data and generates a direction decision for that

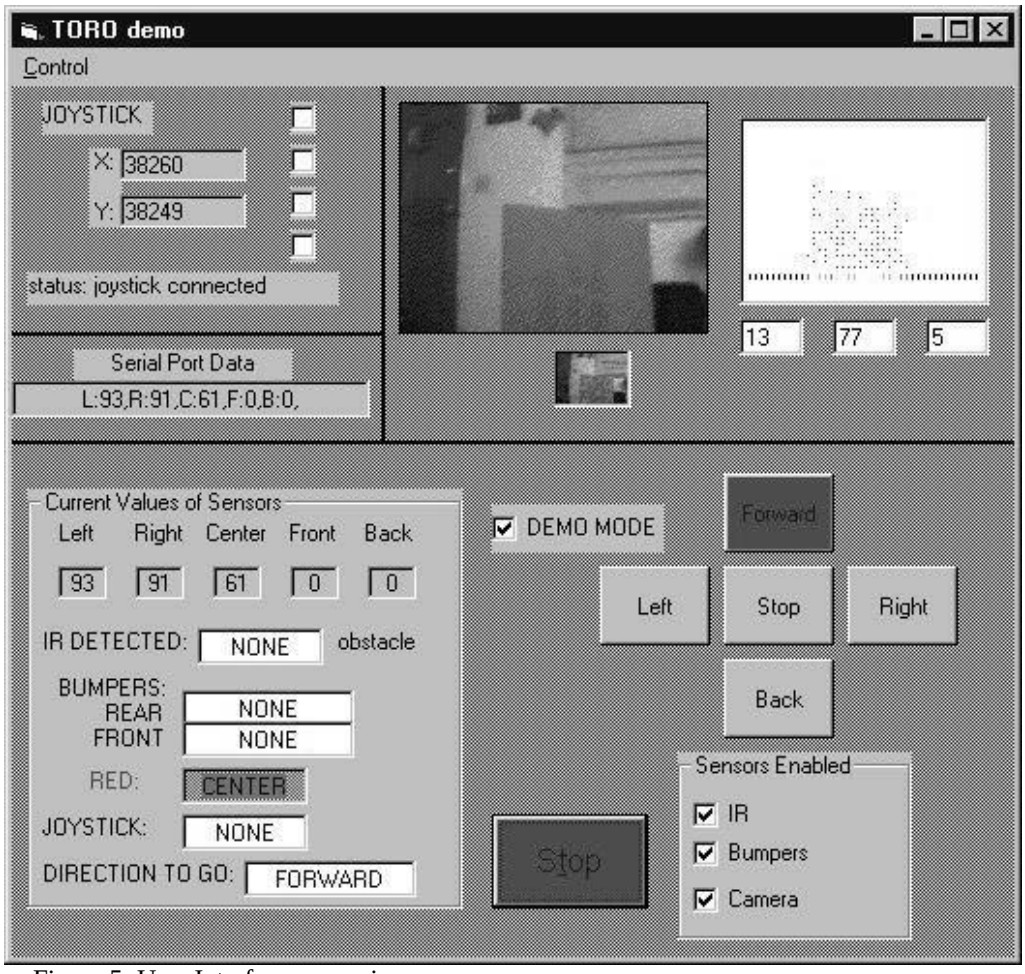

Figure 5: User Interface screen image

sensor. Each sensor has a default direction of either Halt or Forward. There is then a direction arbitrator that generates an overall direction decision by maintaining a hierarchy:

1. Joystick
2. Camera
3. Bump
4. IR

This is not a strict hierarchy in that there are exceptions to this system. For example: if the camera is seeing a lot of red while the front bump sensor is pressed then the assumption is that the target has been hit and the robot stops. The default direction is used to see if the sensor requires attention. If the sensor is passing its default direction then it is ignored and the next relevant sensor in the hierarchy is used to determine the direction.

Since this behavior architecture is used, it is very simple to disable a sensor, in which case it is ignored. Each sensor can be disabled from the main program or via the joystick. Unplugging the joystick will disable it as a sensor. The simplest method of seeing each behavior is disabling all other sensors. For example: to demo the robot, I disabled all sensors except the camera. The robot then remains idle until the camera sensor calls for attention, when it has found red, at which point the sensor's behavior assumes control of the overall system behavior. Because of this enabling and disabling of behaviors, there are actually 4! behaviors.

**User Interface Program**

The interface and control program was written in Microsoft Visual Basic. A screen shot of the interface program is shown in Figure 5. This program shows the most current data sampled from all of the sensors. The direction influence from each sensor is displayed along with the overall direction decision. The program can run in either demo or standard mode. Standard mode is more efficient as it does not update the camera data to the screen. The joystick is used to interact with the program while the robot is

moving. Informative sound clips have been associated with all of the program's options to provide confirmation of selections.

The upper right section details the image processing. A simple algorithm for the detection of red objects is used:

1) The images are reduced to a 40×30 image seen directly below the larger image. Pixel-by-pixel image processing is performed on the smaller image.
2) Red is found by a combination of thresholding filters. The filter used is: "If (bytRed > 125 And bytGreen < 100 And bytBlue < 125)" then this is very likely to be a red pixel.
3) It was also helpful to filter out dark colors since they were affecting the algorithm, this is done by "If bytRed + bytGreen + bytBlue < 150" then this pixel is ignored since it is very dark.
4) The count of red pixels in each of the 40 columns is totaled (represented as a histogram at the bottom of the image) and flattened into 3 values. These values represent the amount of red in the right, center, and left of the robot's field of view.

A threshold value of 6 red pixels per image third is used, after the threshold has been met, the desired direction is the screen portion with the highest number of red pixels.

As a real-time control program, the performance of this application is a crucial design criterion. The overall performance of this application in controlling and communicating with the robot is found by running the control loop 50 times and timing its duration:
Demo mode: 50 times in 32 seconds - 1.56 Hz
Standard mode: 50 times in 15 seconds - 3.33 Hz

**Future Work**

TORO has accomplished many things. I was able to perform near real-time image processing for navigation purposes with an unusual set of tools: a $40 camera, a crumbling laptop, and a Visual Basic control program. The biggest limitation of the robot is the performance of the control program. There are many other avenues that can be explored for improving the code and the overall performance while still keeping the framework currently used. With few adjustments the 3.33 Hz rate could be improved to about 5 Hz. By using other methods of image processing this rate could nearly be doubled. This is important so that smooth real-time navigation decisions can be made. As the robot is currently implemented, if TORO sees a red object in the right of his field of view, he turns right, and by the time he takes the next snap shot he has passed the red object. Compensating for this effect greatly complicates the program. The other limitation is the narrow field of view of the camera. This also leads to the same pattern of behavior described above.

I discovered that doubling the types of sensors used does not result in a doubling of code, but actually a quadrupling of code since the interactions between the data must be accounted for. For this reason it is very difficult to have consistent intelligent behavior from so many sensor systems without programming for very complex interactions. Because of this I feel the platform is seriously under-utilized and could be used for much more interesting work, expanding on the simple hierarchy for behavior generation.

TORO is able to accomplish many advanced features without the costs traditionally associated with these abilities. Since TORO uses a complete PC implementation, the platform can be easily extended. Anything that can be done with a PC (and done quite easily) can now be done on the robot. This makes expanding the robot simple, as demonstrated by the ease with which a camera can be added. The color following behavior has been used to successfully validate TORO as a "proof of concept" platform capable of much more complex behavior interactions.

**References**


Robotics Reference Books:
Fred Martin, The 6.270 Robot Builder's Guide, MIT Media Lab, Cambridge, MA, 1992
Joseph Jones & Anita Flynn, Mobile Robots: Inspiration to Implementation, A.K. Peters Publishers, Wellesley, MA, 1993

Visual Basic Books:
Steven Holzner, Visual Basic 6 Black Book, The Coriolis Group, 1998

Video For Windows Programming:
http://ej.bantz.com, http://www.microsoft.com, http://i.am/shrinkwrapvb

Joystick Prgramming:
http://www.microsoft.com