

SERVO AND SENSOR CONTROL ON SMALL MOBILE PLATFORMS

Jorge Blanch and Sabri Tosunoglu

Florida International University
Department of Mechanical Engineering
10555 West Flagler Street
Miami, Florida 33174

ABSTRACT

System integration is an unavoidable and important part of a project. Considerable amounts of time and resources are always devoted to make sure that all the components in a project work not only properly, but also work together properly. This work discusses how different elements of a “desktop” robotic platform that was designed and constructed at FIU had to be controlled to achieve an effective and mechanically stable platform. For this purpose, we review the platform design, sensor utilization and servo control developed for this project. We then demonstrate that a proper servo and sensor control structure can indeed improve the platform performance appreciably.

INTRODUCTION

Robots relate to their environment via sensors and actuators. It therefore follows that any short-coming of either sensors or servos will severely affect the capabilities of a robot. Besides the almost inevitable noise errors and other hitches inherent to electrical circuits subject to dynamic loads, there are other physical limitations that affect both servos and actuators. Knowing the exact position of a vehicle is a fundamental problem in mobile robotics. In search for a solution, researchers and engineers have developed a variety of systems, sensors, and techniques for mobile robot positioning; yet still there is no truly elegant solution for the problem. The many partial solutions can be categorized into two groups: relative and absolute position measurements. Because of the lack of a single good method, two methods, one from each group, are usually combined to provide reliable positioning [1]. Relative position measurements (Odometry and Inertial Navigation) are derived from robot internal sensors alone and have an uncertainty error associated to them; furthermore, this error propagates and becomes larger over time [2,3]. Absolute positioning methods can be used to reduce the error value so that a more accurate

robot position can be determined. Absolute positioning methods include Magnetic Compasses, Active Beacons, Global Positioning Systems, Landmark Navigation and Model Matching (or Map Matching) [1].

Determining the odometry errors of a mobile robot is very important (no use of inertial navigation in our platform), both in order to reduce them, and to know the accuracy of the position estimated by dead reckoning. The odometry error contains both systematic and nonsystematic components. Systematic errors are those inherent to the platform, they relate to the physical properties of the platform and, to some extent, its relationship with the environment (for example different wheel diameter for each servo). Nonsystematic errors depend heavily on the environment and include unforeseen errors like wheel slippage. Because the platform being tested is for indoor use in a controlled environment, nonsystematic errors should be almost non-existent, so they are assumed to be zero.

In a configuration where there is no off-platform assistance (active beacons and positioning systems are not available), the robot must solely rely on its own sensors to navigate in unknown terrains and, to the extent possible, to compensate for odometry errors. Although techniques like Map-making, Landmark Navigation and Map Matching are all heavily software dependant and therefore not covered in this article, they do require reliable sensor input. An extensive sensor package that can provide large amounts of relevant information is always desirable but often unavailable. When dealing with limited sensor inputs, smart utilization becomes paramount. Since an array of range sensors was not available due to budget constraints, sensors were mounted on servos so they could take readings in almost any direction.

Next, we will briefly review the platform that was developed for this project. Followed by a more

detailed review of the main components such as the servos and sensors and how they are controlled to make this an effective and mechanically stable platform. Finally, some conclusions will be drawn as to the quality and precision of a platform built with off the shelf components meant for hobbyists.

PLATFORM

The platform used in this case is HANCOR (Handheld-Controlled Rover), a small rover created at Florida International University to test the viability of using handheld devices as platform controllers. The platform is based on a Pontech SV203 servo controller board that uses Sharp GP2D120 infrared ranger sensors to detect obstacles and Futaba-type servomotors (servos used in radio-controlled model airplanes, cars, etc.) to drive the wheels. An on-board Palm III handheld controls the platform [4,5,6].



Figure 1. HANCOR platform (handheld connected but not mounted).

The servo controller board is based on a PIC16C73 microchip; it accepts serial data from a host computer (replaced by the Palm III handheld in this case) and outputs a PWM (Pulse Width Modulated) signal to control up to eight RC servomotors [10]. Two servos, modified for continuous rotation, provide power for the driving wheels. Besides servos, the board can control other digital devices that require an on/off signal to be activated. Up to 5 sensors can be connected to the A/D input header, which reads analog voltages between 0 and 5 Volts. The sensor package consists of two infrared range sensors and one digital compass. The digital compass has a resolution of $\pm 22^\circ$ (it detects N, NE, E, SE, etc.), so it cannot be used as a reliable sensor input, but can later be used as aid during map matching [7]. The IR sensors are mounted on two small servos so that they can pan to any direction in a 135° forward-looking field of view

SERVOS

Small budget platforms like the HANCOR often replace expensive servos and position encoders with off-the shelf radio controlled servomotors. RC servos are cheap, easy to control, come in a convenient form factor and are available in different sizes, speeds and power ratings. Servomotors are generally employed for position control; they use a potentiometer as feedback to determine their position. The servo compares its current position to an input PWM signal, and then moves until its position matches the input signal. The “width” of the signal pulse may last anywhere from 0.6 milliseconds (minimum position), to 2.4 ms (maximum position). The Pulse is repeated every 14 to 20 milliseconds (figure 2) [9]. Regular servomotors are used for the positioning of the range sensors, but controlling the driving servos require some modifications.

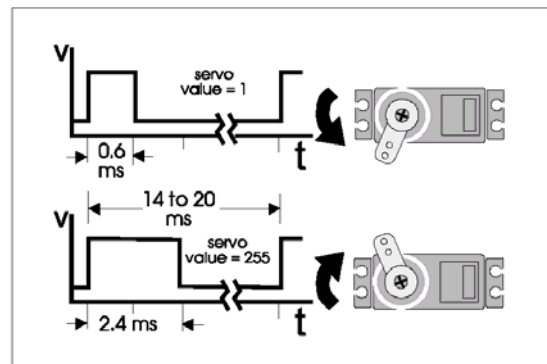


Figure 2. Futaba-type servo timing diagram.

Futaba-type Servomotors were not developed for continuous rotation. In order to utilize servos in situations that require continuous rotation they have to be modified. Servos were not designed for velocity control either. The velocity of a servo depends on how far it is from the desired position. The servo will spin at top speed until it gets close to the desired value, then it will quickly slow down to avoid over-shooting the target position. Usually, continuous rotation and velocity control go hand in hand. Servos that have been “hacked” for continuous rotation have the feedback potentiometer de-coupled from the output gear and set to a constant value (for example 90°). The velocity is controlled by giving the servo a target position (0 to 180°), but since the feedback has been fixed at 90° , the servo will spin forever towards the target position. The further the target position from the fixed position (90°), the faster the servo will spin.

The Pontech SV203 controls its servos via an 8-bit signal (PWM). An 8-bit signal can take 2^8 values (from 0 to 255). If we define the 0 signal as 0° and

the 255 signal as 180°, then the 90° signal will be 128. As can be seen from the data in figure 3, the servo quickly reaches top speed, which leaves us with a relatively small range of useful bit values (roughly from 90 to 170 in our case). Values outside this range will not result in a significant speed variation.

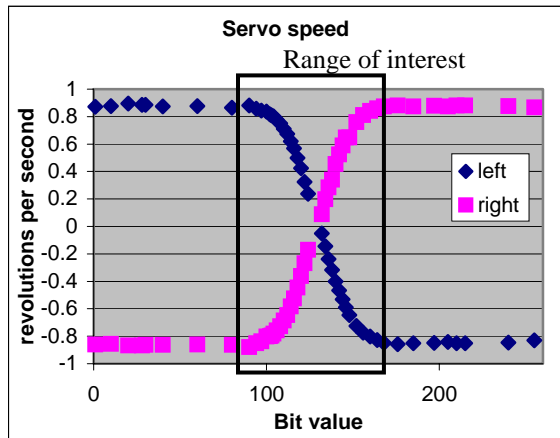


Figure 3. Servo speed measurements.

Servo speeds were measured (in revolutions per second) at different bit signals and stored in Excel (figure 3). Data was then transferred to Matlab to perform some numerical analyses on it; the objective being to find an equation that will return the bit value needed to achieve a desired rotational velocity.

To effectively approximate the data, it first had to be transformed and formatted. Two processes were required. First, the order of the data pairs was changed so that velocity would be our independent variable (input) and the calculated bit our dependent variable (output). Secondly, only values in the range of interest were used; this helped to generate a tighter-fitting curve, since only relevant data was being used (figure 4).

The 3rd degree polynomial approximation [13] that fit the data (from about -0.9 to 0.9) as given by Matlab was:

$$\text{Right bit}(v) = 31.224v^3 + 0.451v^2 + 15.72v + 128.948$$

$$\text{Left bit}(v) = -33.272v^3 + 1.504v^2 - 15.294v + 130.273$$

The 3rd degree approximation looked promising but when tested on the platform it did not perform as well as expected; the platform could not describe a straight path. This approximation is not a method that is accurate enough to drive the platform.

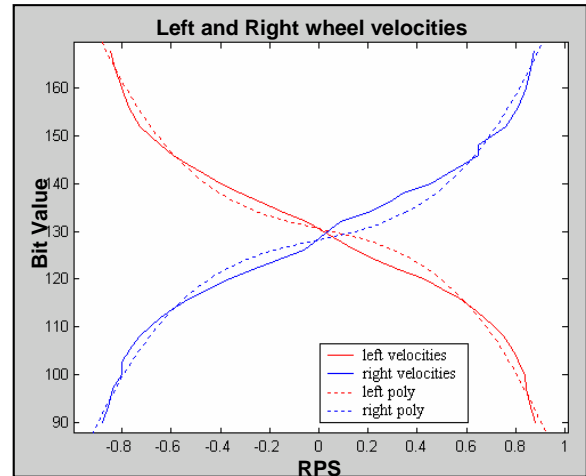


Figure 4. Actual values and 3rd degree polynomial approximations.

Another curve-fitting method that can better approximate the data is the use of splines [13]. Splines are polynomial approximations that are used to fit sub-regions of the data. Since the section of the curve to be approximated is smaller, a much higher degree of accuracy can be achieved.

Data points were then chosen subjectively to break up the curves in sections that presented similar behavior (curvature) (table 1). A simple linear approximation from one point to the next (splines of 1st degree) provided a much better fit to the original data (figure 5) than the 3rd degree polynomials calculated previously. Of course, this came at the cost of having 9 linear equations for each 3rd degree polynomial.

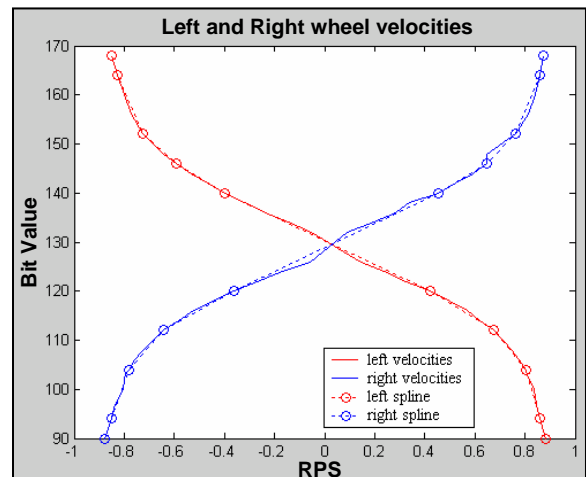


Figure 5. Close-up of range of interest: Gathered data with extracted points superimposed.

Table 1. Data extracted from range of interest.

	Left	Right
bit	rps	rps
90	0.880282	-0.87873
94	0.857633	-0.85106
104	0.802568	-0.78125
112	0.673401	-0.64516
120	0.423729	-0.36058
140	-0.4	0.454545
146	-0.59032	0.648508
152	-0.72464	0.761035
164	-0.82781	0.859107
168	-0.8489	0.874126

Two observations that persuade against searching for more accurate approximations are: 1, the experimental measurements are subject to error; depending on the accuracy of the reading or even on the battery charge, values will vary slightly from one run to the next. And 2, bit commands are integers; therefore there will be gaps in the velocity curves for values that could never be reached because they correspond to a bit value between two integers. On the other hand, the closeness of the spline approximation to the actual data showed that selecting a few finite points can offer sufficient control over the velocity. Not using an equation (or several) to find the bit value corresponding to a desired velocity, results in a crude control over speed values; but, by finding the bit values that correspond to specific velocities (0.8, 0.6, 0.4, 0.2, 0.0, -0.2, -0.4, -0.6 and -0.8 rps, per se), the velocity of the platform can be properly controlled for its full range of values.

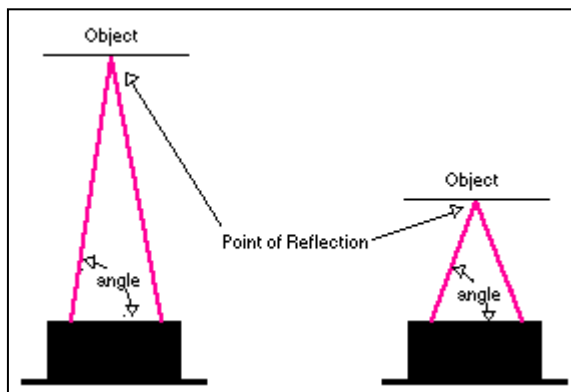


Figure 6. Distance and angle calculation.

SENSORS

The main sensor used by the HANCOR is the Sharp GP2D12 analog infrared ranger [10]. Sharp IR rangers use triangulation and a small linear CCD array to compute the distance or presence of objects

in the field of view (Figure 6), which results in greater reliability and accuracy than many IR sensors that use time-of-flight techniques. Also, these new rangers offer much better immunity to ambient lighting conditions and to the color of the reflected surface than other IR sensors [11]. These sensors have a minimum range of 10 cm (~ 4in) and a maximum range of 80 cm. The beam is very tight, just about 3 cm wide and less than 3 cm in height at 40 cm. Such characteristics make the sensors quite suitable for unidirectional measurements, but not so great for general obstacle detection. Some advantages over ultrasound ranger sensors, traditionally used for obstacle detection, are: IR sensors do not suffer from ghost images; furthermore, the angle at which they face an obstacle can be as high as 60° without affecting distance reading [11]. IR sensors also have much lower power requirement compared to the battery-hungry ultrasound sensors. Finally, price becomes an issue when ultrasound sensors are over five times more expensive than IR sensors. Of course, sonars will always be great detection sensors thanks to their range (2 to 120 cm) and their wider detection area.

Table 2: IR range values for given distances

Distance (cm)	Sensor1	Sensor2
10	239	235
12	210	204
14	185	177
16	169	161
18	152	141
20	139	132
22	129	120
24	120	113
26	113	105
28	108	99
30	101	91
32	95	87
34	87	78
36	81	79
38	79	71
40	74	67
42	69	63
44	65	59
46	61	55
48	57	52
50	54	48
52	50	43
54	48	41

The analog output of the infrared does not change linearly with the distance being measured; table 2 shows the readings taken from the servos from 10 to

50 cm. To find a numerical approximation that can yield a governing equation, the average of sensor 1 and sensor 2 was found and then analyzed.

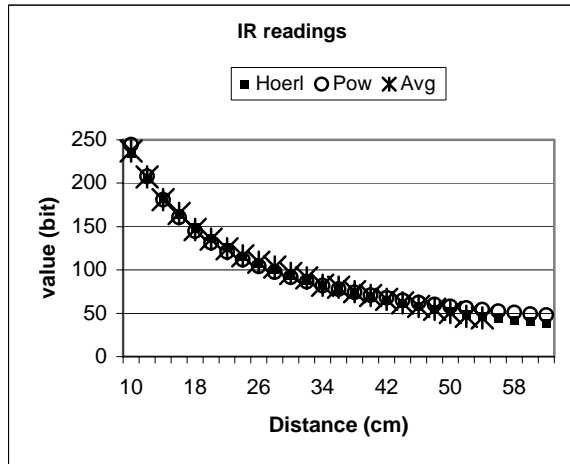


Figure 7. IR distance approximations.

The average values can be approximated by the following power equation:

$$V(d) = 1917.709 * d^{(-0.8941)}$$

which is very similar to that of [11]

$$V(d) = 2141.72055 * d^{(-1.078867)}$$

A better approximation can be found using a Hoerl Model, which is a modified power equation: $V(d) = 997.82 * 0.985^{d * (-0.563)}$, but it is not really necessary to perform the extra power function for a marginal increase in accuracy.

To turn IR range sensors into effective obstacle detectors, the servos must reposition the sensors to get several readings of the terrain ahead of the robot. If the sensors have to be reoriented, timing becomes an issue. Several factors have to be taken into account to determine the frequency at which readings can be taken. First of all, the refresh rate of the IR sensors determines the maximum frequency at which readings can be taken. The velocity of the servos dictate how long it will take to reach the desired orientation. The response of the controller board indicates how long it takes for the board to poll the AD header for a new reading and return it to the handheld. The handheld connects to the controller board at a certain speed. Finally, the Palm has to perform many computations, and can only read from the serial port every so often.

The IR sensors continuously takes distance readings every 38 ms, for a total of about 25 readings every second (25Hz)[10]. The Hitec ht-85 servos have a nominal operating speed of 0.11sec/60° at

4.8V. Even though the speed of the servo is proportional to the displacement covered, the nominal speed can be used to approximate the time it takes the servo to move a certain arc. At 9600 baud, which is the standard connection speed for the controller board, it takes 4ms to transmit a simple output command of 4 bytes. For more complex commands it takes the Palm 25ms to transmit the command and receive the result as ASCII data. After the data is received, it must then be converted to an integer for it to be useful. In practice, is rare to get sensor readings faster than 50ms. It is possible to connect the controller board to transmit at up to 38400 baud, which should reduce considerably the communication delays, but since the servo response is the main cause for delays in the architecture, it is not necessary.

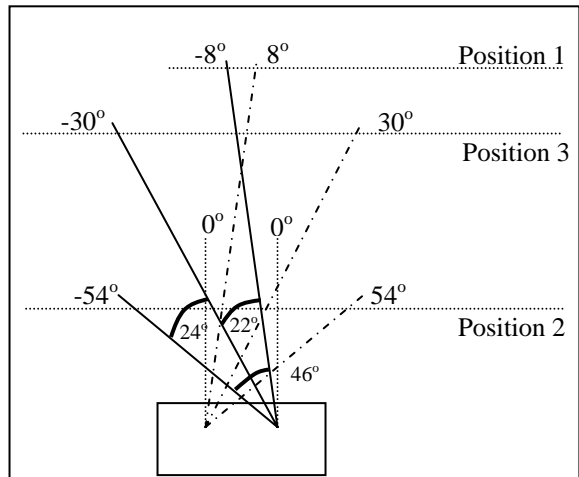


Figure 8. Obstacle detection.

The GP2D12 takes continuous readings, but the Palm takes discrete readings individually. In order to effectively cover the front area of the rover, each sensor takes three readings as seen in figure 8. Given the Hitec's nominal velocity, the different times it takes a servo to go from one position to the next are: Position 1 to Position 2: 46° ~ 0.084sec; Position 2 to Position 3: 24° ~ 0.044sec; Position 3 to Position 1: 22° ~ 0.04sec. Since the IR ranger takes 38ms to refresh, the robot should wait that much longer before reading from the sensor. Therefore, the time delay before a reliable reading can be taken is at least:

$$\text{To position 1: } 0.040 + 0.038 = 0.079\text{sec}$$

$$\text{To position 2: } 0.084 + 0.038 = 0.122\text{sec}$$

$$\text{To position 3: } 0.044 + 0.038 = 0.082\text{sec}$$

Unfortunately, PalmOS does not have a wait() or delay() function, to wait long enough before taking the reading. Fortunately, Palm does have a

getTicks() and TicksInASecond() functions. With these two an approximate waiting function can be written:

```
void WaitMS (int miliseocs)
{
    UInt32 zero, ticks, time;
    zero = TimGetTicks( );
    ticks = zero +1;
    //convert time from ms to ticks
    time = (miliseocs / 1000.0) *
    SysTicksPerSecond( );
    while(ticks - zero <= time)
        ticks= TimGetTicks( );
}
```

This function will do nothing other than check the current “tick” until the predetermined value (obtained from milliseconds) is reached. I should be noted that there are 100 ticks/ sec when running on a Palm, which means that the function is only accurate to 10’s of millisecond.

As it stands, the algorithm is the following:

- ⇒ OS awaits for input (15ms)
- ⇒ no input
- ⇒ if running autonomous
- ⇒ Sweep [
 - ⇒ move to Position 1
 - ⇒ waitMS(90) => read IRs
 - ⇒ move to Position 2
 - ⇒ waitMS(130) => read IRs
 - ⇒ move to Position 3
 - ⇒ waitMS(90) => read IRs]
 - ⇒ collision avoidance [
 - ⇒ change motion (if necessary)]
 - ⇒ increment counter.

This means that the HANDCOR needs at least $0.015 + 0.079 + 0.122 + 0.082 = 298\text{ms}$ to gather the readings for the obstacle avoidance subroutine. Because there are two forward looking sensors and each has three positions, if an object appears immediately after a reading is taken, it will take one full cycle before that object is detected. A full cycle is 298 ms plus the time it takes the palm to run any internal functions (this varies, but is generally less than 50 ms). Each sweep pattern covers just over 45 degrees, and they complement each other to provide reasonable detection of obstacles ahead of the robot. When the obstacles are closer, the overlapping fields provide even better detection. Given that the effective range of the sensor is just over 40 cm (reliability starts failing after about 50 cm), the sensor

refresh rate limits the forward speed of the robot to 40 cm in 300 ms, or it would be possible to hit an object before seeing it. The driving servos’ maximum speed is under 0.89 revolutions per second; with a wheel diameter of 7.6cm (3in), the maximum velocity the robot can achieve is under 24 cm in 1 second, which is well within safe range.

And advantage to having both sensors mounted on servos is that they can be directed, and they can be combined for greater accuracy. By using trigonometry [12], two readings can be taken from the same obstacle and then compared to determine its distance with greater accuracy. If a sensor detects an obstacle at a distance d_R when its orientation is θ_R <1> (figure 9) it is very simple to convert it to coordinates relative to the robot < 2 >, and then to coordinates relative to the other sensor < 3 >.

$$\vec{d}_R = \begin{bmatrix} d_{RX} \\ d_{RY} \end{bmatrix} = \begin{bmatrix} d \cdot \cos(\theta_R) \\ d \cdot \sin(\theta_R) \end{bmatrix} \quad < 1 >$$

$$\vec{D} = \begin{bmatrix} D_X \\ D_Y \end{bmatrix} = \begin{bmatrix} U_{RX} \\ U_{RY} \end{bmatrix} + \begin{bmatrix} d_{RX} \\ d_{RY} \end{bmatrix} \quad < 2 >$$

$$\vec{d}_L = \begin{bmatrix} d_{LX} \\ d_{LY} \end{bmatrix} = \begin{bmatrix} D_X \\ D_Y \end{bmatrix} - \begin{bmatrix} U_{LX} \\ U_{LY} \end{bmatrix} \quad < 3 >$$

Finally, θ_L is

$$\theta_L = \arctan\left(\frac{d_{LY}}{d_{LX}}\right) \quad < 4 >$$

From the platform, $U_R = U_L = 4$ cm and the angles $\phi_R = 35^\circ$ $\phi_L = -35^\circ$. Back substituting into < 4 >

$$\begin{bmatrix} d_{LX} \\ d_{LY} \end{bmatrix} = \begin{bmatrix} U_{RX} + d_{RX} - U_{LX} \\ U_{RY} + d_{RY} - U_{LY} \end{bmatrix}$$

$$\theta_L = \arctan\left(\frac{4 \cdot \sin(35) + d \cdot \sin(\theta_R) - 4 \cdot \sin(-35)}{4 \cdot \cos(35) + d \cdot \cos(\theta_R) - 4 \cdot \cos(-35)}\right)$$

Assuming that there is no other obstacle between the second sensor and the original obstacle, θ_L can be used to get a second reading, d_L from the obstacle, which can then be transformed to a second distance reading D . The redundant distance readings should result in a more accurate reading when compiling maps.

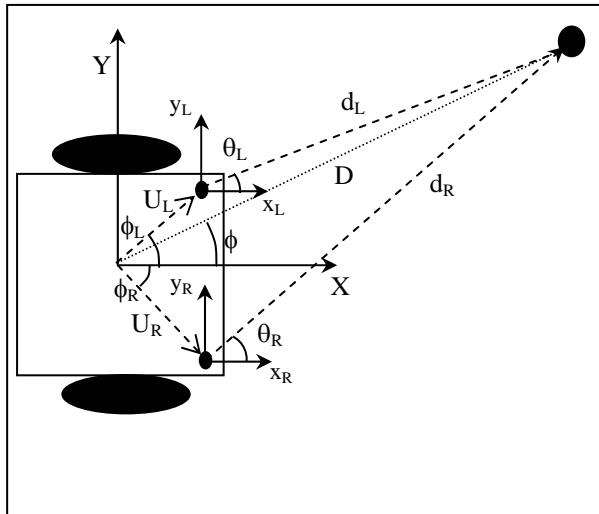


Figure 9. Obstacle detection.

CONCLUSIONS

System integration is indeed a difficult and time consuming task, especially when components have loose tolerances that can only be overcome by combining hardware and software modifications.

Radio-controlled Servos are a low cost alternative to steppers and servos, but they do not allow for fine control of position or motion and therefore should not be used in places where precision is critical. RC servos worked great to orient the sensors, albeit not very quick for autonomous running, they made it very simple to face the sensors in any desired direction, on the other hand, using RC servos to drive the rover, resulted in a difficult to control platform

Infrared rangefinders worked consistently and could deliver data extremely fast, but they had to be timed so that the servo would reach the desired position before readings were taken.

One source of errors that could not be overcome was the electrical noise. This noise was caused by the dynamic electric loads generated by the servos, and inability of the platform to filter them out. It should also be noted that during testing, the time it took for the WaitMS() function to end varied widely (this is not a reliably consistent delay method).

REFERENCES

- [1] Borenstein J. and Feng L., "Measurement and correction of systematic odometry errors in mobile robots," *IEEE Trans. Robot. Automat.*, vol. 12, pp. 869-880, Dec. 1996.
- [2] Martinelli A., "The Odometry Error of a Mobile Robot With a Synchronous Drive System" *IEEE*

- transactions on robotics and automation, vol. 18, no. 3, June 2002.
- [3] Kelly A., "General solution for linearized systematic error propagation in vehicle odometry," in *Proc. Int. Conf. Intelligent Robot and Systems (IROS01)*, Maui, HI, Oct. 29-Nov. 3 2001, pp. 1938-1945.
- [4] Blanch J., and Tosunoglu S., "Hand-Held Computers as Mobile Platform Controllers," *Florida Conference on Recent Advances in Robotics*, Florida State University, Tallahassee, Florida, May 10-11, 2001.
- [5] Blanch J., and Tosunoglu S., "Enhanced Small Mobile Platforms Controlled by Hand-Held Computers," *IASTED International Conference on Robotics and Application (RA 2001)*, Tampa, Florida, November 19-22, 2001.
- [6] Blanch J., and Tosunoglu S., "Control of Mobile Platforms via Hand-Held PDA's," *the 15th Florida Conference on Recent Advances in Robotics*, Florida International University, Miami, Florida, May 23-24, 2002.
- [7] Varveropoulos V., "Robot Localization and Map Construction Using Sonar Data." The Rossum Project, available at <http://rosum.sourceforge.net/> accessed on December 2002.
- [8] "Futaba Servo Motors," accessed from www.futaba-rc.com/servos/futm0029.html, accessed on October 2002.
- [9] Pontech SV203 Servo motor controller board user manual. Available online at www.pontech.com. Accessed on December 2002.
- [10] "Acroname Articles Demystifying the Sharp IR Rangefinders," accessed from www.acroname.com/robotics/info/articles/sharp/sharp.html. Accessed on October 2002.
- [11] Technical info for the PPRK <http://www-2.cs.cmu.edu/~pprk/> Accessed on January 2003.
- [12] Leon, S. J. "Linear Algebra with Applications." Upper Saddle River, New Jersey. Prentice Hall, Inc. 1998.
- [13] Gerald C. F., and Wheatley P. P., "Applied Numerical Analysis." Reading, Massachusetts. Addison Wesley Longman, Inc. 1999.