Achieving Consistency in Practical Robotics
Thomas MacWilliam
Malden Catholic High School
tmacw09@gmail.com

Achieving Consistency in Practical Robotics

1 The Necessity of Consistency

A robot is defined as "a machine that resembles a human and does mechanical, routine tasks on command." [1] Often, robots are designed to perform a complex or repetitive task with precision and efficiency outside the realm of human capacity. In tasks ranging from assembly-line production to brain surgery, a robot must be able to reliably complete a specified task regularly. Subsequently, robots must be designed with consistency in mind, such that a "routine task" can be completed in a habitually congruous manner, lest their very purpose be lost in the wake of disparity.

1.1 The Illusion of Consistency

In dealing with robots, especially those constructed of Lego™ parts, a motor cannot be expected to spin a precise distance on a regular basis. For example, consider the following command, which will spin MOTOR_LEFT 1,000 ticks at a speed of 500:

```
mrp(MOTOR_LEFT,500,1000L);
```

Ideally, this motor would spin the same distance at the same speed each time this command was executed, as this simple line would seem to be an accurate way to drive a calculated distance. Yet, this is a practical impossibility. This fundamental command does not take into account gear slipping or the battery strength of the robot, and will therefore not yield the same result each time it is executed. For example, a gear slip on one run could cause the robot to actually move 1,100 ticks, while a jam on the next run could cause the motor to tangibly spin 900 ticks. As a result, a motor cannot reliably be spun a certain distance using merely one line of code. The issue of precision, particularly in dealing with motors to drive the robot to a particular location, is therefore more intricate, and can only be achieved with the use of sensors to ensure a robot "knows" its position.

2 Mechanical Consistency

First and foremost, any robot must have the foundation of a solid mechanical design. Motors and servos must be firmly mounted at all points throughout the robot so they maintain a constant position while the robot is in motion. Trogdor, one of our robots, uses a lifting arm in order to stack cups on top of one another (Figure 1). The success of our cup-stacking mechanism depends on the constant position of both the servos on which the arm rotates as well as the bed into which the cups are released. If the cup bed were to shift to the left or the right, the arm would not properly store the cups, and they would easily fall out as the robot was driving. Or, if

the mount for the arm were to shift backwards from the momentum of the swing, the cups would be released behind the robot, landing back on the board. Therefore, the importance of a rugged motor or servo mount cannot be undermined, for a greater degree of consistency can be achieved using a motor with limited mobility inside its mount.

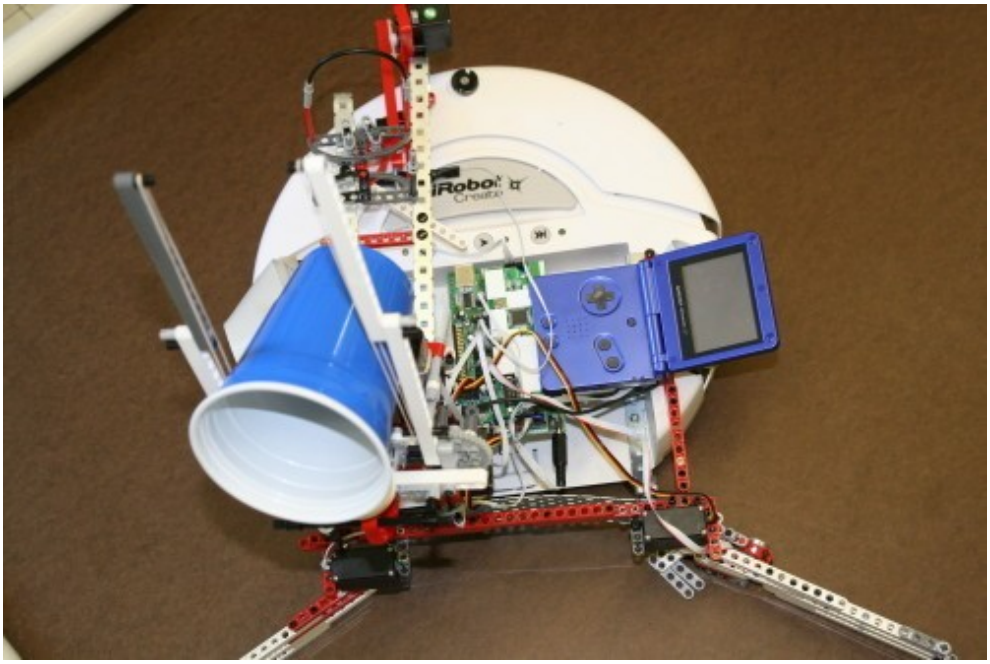

Figure 1: Trogdor's Cup Stacker

2.1 Geartrain Consistency

Direct-drive robots, in which wheels are connected directly to motors, are frequently inconsistent. This overly simplistic wheel mount often results in wheel slipping and increased tension on the motor itself, which could reduce its efficiency. Instead, robots that drive on a geartrain, in which a series of gears connects the motor to the wheel axle, are more consistent. A geartrain greatly reduces the potential for wheel misalignment; on a direct-drive wheel, the axle can easily slide, causing the robot to drive left or right even when programmed to drive straight. Furthermore, a geartrain can be configured to maximize torque or speed, depending on gear positions, in order to prevent the wheel from slipping on the board. Even so, the possibility of gear slipping is inevitable on a geartrain, and gears must be carefully positioned so their teeth line up precisely.

3 Programmatic Consistency

As previously stated, even the ideally constructed mechanical robot cannot be expected to perform a task in precisely the same manner using simple motor commands. Instead, a combination of digital and analog sensors can be used to plan out a specific path for the robot, utilizing a task-oriented loop structure in which the robot searches for predetermined values before proceeding to the next goal.

3.1 Landmark Positioning

Landmarks, or specific objects at a predetermined, fixed position, can be utilized in tracking the robot's position on a board. In this year's game, landmarks include the plant and crew piles, the satellites, and the pipe edges of the game board. Both digital and analog sensors can be used in landmark positioning, in which the robot moves until a specific sensor value is detected. For example, a touch sensor can be used to determine the distance from a pipe; the robot can drive forward until the touch sensor is tripped (and returns a value of 1), then stop. This can be accomplished using the following code:

```
while(digital(FRONT_TOUCH) != 1) {
     mav(LEFT_WHEEL,300);
     mav(RIGHT_WHEEL,300);
}
ao();
```

In this example, both wheels are spun using the move_at_velocity command instead of the previous move_relative_position function, since the commands are inside a loop. An instantaneous gear slip or jam will not severely affect this code sample, as the robot will continue driving until the touch sensor is tripped and therefore end up in the same position regardless of the means. Furthermore, touch sensors are more consistent than analog sensors, as the potential for sensor malfunction is greatly reduced because digital sensors return only two values and trigger based on tangible, physical contact.

Digital sensors, while useful in determining if a robot is touching a particular object, are limited in their utility. Analog sensors, while somewhat less reliable than their digital counterparts, can be used to determine the exact distance from an object even when the robot is not touching it. For example, the following code excerpt utilizes an infrared sensor to spin clockwise until the robot is a set distance from a plant pile:

```
while(analog(FRONT_IR) < 200) {
     mav(LEFT_WHEEL,40);
}
ao();
```

Analog and digital sensors can be combined in order to plan out a specific path for the robot, in which reaching each landmark can be viewed as a separate task. Using the above examples, the robot can drive forward until a touch sensor is tripped by an object such as a pipe, then turn right until a ball pile or cup is found. In planning the path of the robot, the aforementioned sensor limitations must be considered; a touch sensor can reliably tell if a robot is touching a pipe, but the angle of the robot cannot be easily determined. However, a digital sensor can be used to read the distance from another object and consequently calculate the angle of the robot in relation to the pipe. A path that follows a pattern of digital and analog sensor readings is therefore most effective. Consistency can thereby be achieved by orienting the robot's mission around locating certain objects instead of driving fixed distances.

3.2 Proportional Speed

Stopping and starting motors abruptly hinders the consistency of a robot. Attempting to bring a moving robot to a complete stop by immediately stopping both motors is largely ineffective, as the robot will most likely skid or twitch while stopping, thereby changing its traveled distance or angle. While this change in position may appear minimal, the robot can greatly deviate from its intended path after several consecutive skids. In order to limit "bang-bang" control, the speed of motors must be gradually reduced in a process similar to braking. As the robot approaches a specific object, its speed must decrease proportionally to its position. Therefore, analog sensors are most effective in achieving proportional speed. In the following example, the robot will approach an object, decreasing its speed until it reaches the desired distance:

```
while(sonar(FRONT_SONAR) >= 136) {
     int speed = (2 * sonar(FRONT_SONAR)) - 260;
     // speed over 400 is too fast
     if(speed > 400)
          speed = 400;
     // speed under 20 is too slow
     if(speed < 20)
          speed = 20;
     mav(LEFT_MOTOR,speed);
     mav(RIGHT_MOTOR,speed);
}
ao();
```

In this example, the speed of the robot is determined based on its distance from an object, determined by sonar. As the robot approaches the object, the value of the sonar will decrease, and the robot's speed will reduce proportionally, using the equation $2x - 260$. Appropriate adjustments to the speed must also be made so the robot does not move too fast for the sonar to be read or wastes time moving slowly. The above excerpt is more consistent than simply stopping a motor, as the gradual reduction of speed reduces wheel slippage and allows the robot to stop in a constant position.

3.3 Failsafes

The failure of motors, sensors, and digital cameras is seemingly inevitable. As a result, the success of a program must not depend on the ideal functioning of all aspects of the robot. Instead, failsafes must be inserted into the program so the robot can take appropriate action if something goes wrong to salvage as many points as necessary. The severity of failsafes can range from a minor compensation due to a camera failure to a complete abort of the mission due to a sensor malfunction. Minor failsafes should be placed throughout the program so that the robot does not become too reliant on the success of one motor or sensor. For example, our robot's arm uses two infrared sensors on its arm to determine if a cup has entered the claw. Should one of these sensors fail, the other can still be used, and the mission will not have to be abandoned completely. However, in dealing with functions that are integral to the overall

success of the program, a complete abort may be more beneficial. For example, the following code is used to drive forward until a cup enters the robot's claw:

```
while(1) {
     // cup found, so stop looking
     if(analog(ARM_LEFT_IR) < 180 ||
       analog(ARM_RIGHT_IR) < 180) {
          ao();
          break;
     }
     // cup not found, so drive straight
     else {
          mav(LEFT_WHEEL,500);
          mav(RIGHT_WHEEL,500);
     }
}
```

However, this segment is largely inefficient, as the robot will continue to drive forward if the cup is never found. However, integrating a timer into the program so the robot only searches for the cup for a specific amount of time would allow for the implementation of an abort failsafe:

```
int timer;
for(timer = 0; timer < 200; timer++) {
     if(timer > 198) {
          // insert abort code
          // stop robot
          ao();
          while(1) {}
     }
     // cup found, so stop looking
     if(analog(ARM_LEFT_IR) < 180 ||
       analog(ARM_RIGHT_IR) < 180) {
          ao();
          break;
     }
     // cup not found, so drive straight
     else {
          mav(LEFT_WHEEL,500);
          mav(RIGHT_WHEEL,500);
     }
}
```

Using this program, measures to be taken in the event the cup is not found can be inserted. At this point, the mission can be completely aborted, since it would be impossible to complete, and the robot should try only to salvage as many points as necessary.

4 Integration of Mechanical and Programming Consistency

Ultimately, a robot must have sound mechanical and programming design if it is to be successful and consistent. Neither aspect can take priority over the other, and cooperation between these two different fields must be fostered. For example, it may be impossible to mount a sensor such that the distance from an object can be determined at a perpendicular angle. Therefore, the programmer must be able to adjust values to reflect the discrepancies in readings. Or, it may be impossible to read the value of a particular sensor; in this case, mechanical adjustments should be made. Only through this synthesis of software and hardware can a consistent robot be designed.

References

[1] "robot." *Dictionary.com Unabridged (v 1.1)*. Random House, Inc. 03 Jun. 2008. <Dictionary.com http://dictionary.reference.com/browse/robot>.