Lukesky the Walker

A Lego Mindstorm NXT 2.0 Robot

Missouri S&T

Gary Steelman

5/1/2011

Contents

Background

In this project I implement a basic robot using the Lego Mindstorm NXT 2.0. The robot has functionality to move between any two desired real-world points via locomotion produced from two four-bar mechanisms. Since the robot is a walking robot, it was named Lukesky the Walker [5]. There were many implementation details that may not be obvious immediately and are thus explained in detail in this report.

Project Goals

The main goal of this project is to exhibit implementation of the mathematical concepts demonstrated in classroom lecture. These concepts include:

- Euclidian Spatial Descriptions
    - Use of 4x4 matrices called *reference frames* to represent the orientation and location of a 3-Dimensional Coordinate Axis in Euclidian space.
- Coordinate Transformations
    - Pre or post multiplication of reference frames to create chains of reference frames to rotate and translate reference frames.
- Manipulator Kinematics
    - Use of the Denavit-Hartenberg (DH) Convention with reference frames to define the link and joint chains of a system.
- Inverse Kinematics
    - Utilizing trigonometric expressions to read the orientation of a reference.
    - Use of Euler Angles to define the transformation necessary to obtain a specified robotic wrist orientation from another orientation.
- Jacobians
    - Use of a skew symmetric matrix's properties to derive the linear and angular velocity of a reference frame in real time.
- Path Planning and Trajectory
    - Use of potential field path planning wherein obstacles are represented as repulsive fields and targets are represented as attractive fields.
    - Use of quintic polynomials to create paths with smooth transitions from one segment to another.
- Machine Vision
    - Application of filters to images to manipulate the image for mathematical use.
- Cams, Gears, and Mechanisms
    - Use of cams, gears, and mechanisms to achieve repetitive motion in a system while minimizing stress on actuators in the system.

To embody these concepts in a project form, the following goals for a robot were created:

1. The robot would have locomotive ability through walking rather than driving.
2. The robot would seek a specified color source then seek another source of the same color.
3. The robot would avoid obstacles in its path.
4. The robot would return to the place it started searching from after a certain amount of time or certain number of color sources found.

Use of all concepts mentioned would necessitate a large project. Due to time constraints only a partial number of the concepts were implemented.

Project Goals Accomplished

The following concepts listed in the Project Goals section were implemented:

- Euclidian Spatial Descriptions
    - Lukesky tracks his position and orientation using reference frames. Even though Lukesky only has the ability to move in a 2-Dimensional plane, the reference frames used are still 3-Dimensional frames.
- Coordinate Transformations
    - To track his position and orientation, Lukesky multiplies his current position's reference frame with a transformation frame to reach the destination frame.
- Manipulator Kinematics
    - The DH Convention was used to define Lukesky's body. The scope of this project only required that one frame be defined; though theoretically it could be useful for one frame to be defined for the end of each of the six legs.
- Inverse Kinematics
    - For Lukesky to utilize the reference frame system properly, it is necessary to extract the angles off of the world reference frame easily. This is implemented using trigonometric expressions derived specifically for Lukesky. These expressions allow detection of Lukesky's orientation.
- Cams, Gears, and Mechanisms
    - A chain of two gears is used for each motor to decrease the output speed and increase the output torque such that the legs operate smoothly and have enough power to move Lukesky.
    - The leg mechanism is generated from a four-bar design so that the repetitive forward motion can be generated from a single motor.

The following goals were achieved using the concepts implemented:

1. The robot would have locomotive ability through walking rather than driving.
    - The leg mechanism gives Lukesky the ability to walk.
2. The robot would return to the place it started searching from after a certain amount of time or certain number of color sources found.
    - The use of reference frames with coordinate transformations and forward and inverse kinematics allows Lukesky to return to the initial point at any time.

Robot Comopnents

Locomotion Mechanism

Legs

Lukesky is an adaptation of Daniele Benedettelli's NXTAPOD. [1]

*"[The NXTAPOD is a] simple hexapod robot that can walk and reverse-turn using **just one motor**. It can detect obstacles using the ultrasonic sensor."*

The NXTAPOD's locomotive abilities are based on a four-bar mechanism and produce forward motion when the legs are driven exactly one-half motor rotation out of phase. Its locomotive abilities are similar to how four-legged animals walk. By driving legs in opposite positions forward, locomotion is produced. See the figure below for details.

Figure 1 - Daniele Benedetelli's NXTAPOD. [1]

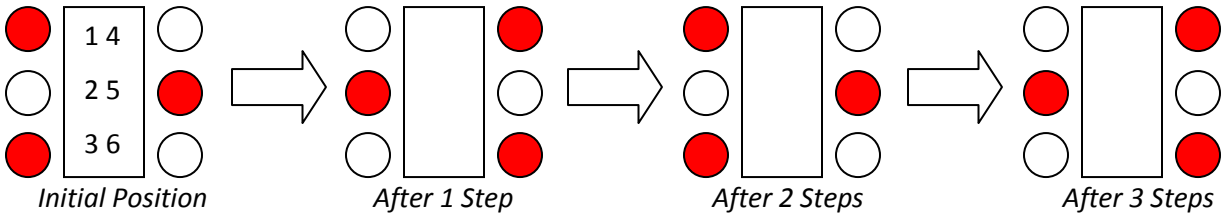

Figure 2 - Leg positions through a few steps of the NXTAPOD.

The red circles indicate which of the NXTAPOD's six legs are in contact with the ground for each step. The legs on the left hand side are numbered 1, 2, 3 and legs on the right hand side are numbered 2, 4, 6. To step, the legs that will contact the ground in the next step are rotated forward or backward and when placed on the ground, lift the legs that previously were on the ground. This cycle is repeated and thus forward or backward locomotion is achieved. This can be mathematically modeled as:

$$S_{k+1} = S - S_k \; where \; S_0 = \{1,3,5\} or \; \{2,4,6\}$$

Where S is the set of all six legs and S_k is the set of legs in contact with the ground this step. At any given time only three of the six legs are in contact with the ground.

Motors

The NXTAPOD has only forward locomotion and rightward rotation ability. It walks forward until it encounters an object, then utilizing a special braking mechanism, rotates to its right and continues to walk forward. This range of motion is inadequate to achieve the Project Goals; the single motor driven design of the NXTAPOD was insufficient. The NXTAPOD design was modified to include the twin-motor system:

$$\text{Twin-Motor System} = \begin{cases} Motor\ B \\ Motor\ A \end{cases}$$

Where *Motor B* drives the left side legs and *Motor A* drives the right side legs. The motors can be driven entirely *independently* and therefore unlike the NXTAPOD, each side of legs can be driven independently. To accommodate the twin-motor system, Lukesky is wider and heavier than the NXTAPOD.

Gears

For smooth locomotion, it was necessary to significantly alter the rotation speed of the NXT 2.0 motors. The motors at 100% of their possible maximum speed moved far too rapidly for the NXTAPOD to follow the Stepping Equation without large amounts of error. The motors were also incapable of producing a satisfactory amount of torque for locomotion. For this reason, the stepping mechanism includes two different gears for each motor:

$$\text{Gear System} = \begin{cases} Gear\ 1 = 12\ teeth \\ Gear\ 2 = 36\ teeth \end{cases}$$

Gear 1 is placed along the axle through each motor and rotates in at a 1:1 rate with the motor. *Gear 2* is placed in contact with *Gear 1*, but as part of a gear chain. The outputs of the Gear System are:

$$\text{Output Torque} = \frac{Gear\ 2}{Gear\ 1} = \frac{36\ teeth}{12\ teeth} = 3 * \text{Motor Torque}$$

$$\text{Output Velocity} = \frac{Gear\ 1}{Gear\ 2} = \frac{12\ teeth}{36\ teeth} = \frac{1}{3} * \text{Motor Velocity}$$

This generates more torque and slower angular velocity for the output of the motors, which was necessary for smooth locomotion.

Vision Mechanisms

Color Sensor

It was intended for the Light Color sensor to be equipped onto Lukesky, however, due to time constraints, no behavioral algorithm was developed to use the input from this sensor. The sensor was thus omitted from the final design.

Ultrasonic Sensor

As with the NXTAPOD, Lukesky is equipped with the NXT 2.0 Ultrasonic Sensor. This sensor provides Lukesky with the ability to detect objects directly in front of him in real time.

Implementation Methodology

3D Library

Features

To accurately track Lukesky's position in space, a reusable 3D reference frame library was created using the NXC programming language [2] and the BricxCC IDE [3]. The NXC programming language is a language that aims to emulate C for the Lego Mindstorm NXT 2.0. The most useful features of this library are listed here:

- Create a 3D reference frame.
- Translate a 3D reference frame.
    - Relative to Lukesky's current position.
    - Relative to the constant world frame.
- Rotate a 3D reference frame.
    - About any axis, X, Y, or Z.
    - Relative to Lukesky's current orientation.
    - Relative to the constant world frame.
- Multiply two reference frames.
- Calculate the distance between two reference frames.
- Output a reference frame to the LCD screen of the NXT 2.0 Brick.

Definitions

A 3D reference frame, A, is defined as:

$$A = \begin{bmatrix} X_x & Y_x & Z_x & x \\ X_y & Y_y & Z_y & y \\ X_z & Y_z & Z_z & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where the first column, represented as $X_{<axis>}$, is the X axis of the reference frame represented as a vector in three components, each in the direction of the world X, Y, and Z axes. The second column represents the reference frame's Y axis in its three components and the third column represents the reference frame's Z axis in its three components. The fourth column represents the location (x, y, z) relative to the world reference frame. The world reference frame is defined as:

$$\text{World Reference Frame} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

As can be seen, the world reference frame essentially represents the *origin* and the direction of each of the normalized X, Y, and Z axes for the world.

To accomplish a rotation or a translation, the library essentially multiplies a reference frame by the appropriate rotation or translation frame.

The translation reference frames are:

$$T_{X,a} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};\ T_{Y,b} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};\ T_{Z,c} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where $T_{X,a}$ means a translation along the reference frame's X-axis by a number a. The translations for the Y and Z axes are defined analogously.

The rotation reference frames are:

$$R_{X,\alpha} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_\alpha & -s_\alpha & 0 \\ 0 & s_\alpha & c_\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};\ R_{Y,\beta} = \begin{bmatrix} c_\beta & 0 & s_\beta & 0 \\ 0 & 1 & 0 & 0 \\ -s_\beta & 0 & c_\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};\ R_{Z,\gamma} = \begin{bmatrix} c_\gamma & -s_\gamma & 0 & 0 \\ s_\gamma & c_\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where $R_{X,\alpha}$ means a rotation about the reference frame's X-axis by a *radian* value α and c_α represents the cosine of the angle alpha and s_α represents the sin of the angle alpha.

Optimizations Specific to NXC

In general, a 3D reference frame is a 4x4 matrix. However, for the scope of this project the fourth row of the reference frame is essentially useless as it is *always* $[0 \quad 0 \quad 0 \quad 1]$. To reduce memory consumption by sixteen bytes per frame and reduce the number of operations per translation or rotation, the last row of the reference frame was omitted. This change is visualized as thus:

$$\begin{bmatrix} X_x & Y_x & Z_x & x \\ X_y & Y_y & Z_y & y \\ X_z & Y_z & Z_z & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} X_x & Y_x & Z_x & x \\ X_y & Y_y & Z_y & y \\ X_z & Y_z & Z_z & z \end{bmatrix}$$

Additionally, to improve library performance for the NXC language, multiple optimizations were implemented that would not necessarily be more efficient for a standard C implementation:

- For each operation that required accessing an index of a reference frame, ie, `A[0]`, the elements were accessed only once and their values stored in a local temporary variable, ie, `A0`.
- For each trigonometric operation, ie, $\sin(\theta)$, the value was stored in a local temporary variable, ie, `stheta`. Since the rotation frames utilize the negation of trigonometric functions, ie, $-\sin(\theta)$, having the value of $\sin(\theta)$ stored locally allowed omission of another call to the sin function just by storing `-stheta`.
- Storing the values for the trigonometric functions not only allowed less calls to calculate trigonometric values, but also allowed use of the NXC function `ArrayBuild()`. This function is the fastest way to create or store multiple values in an array.
- Simple functions, like initialization or copying of a reference frame were defined using the `#define` preprocessor macro to avoid additional function calls on the stack.

- Each time a reference frame was set equal to another reference frame, ie, `A=B`, the assignment was made exactly as `A=B`. Assignment in this fashion generates a single MOV assembly call. This style of assignment is different from standard C where a `memcpy()` call would have to be used.
- Each time a reference frame is rotated (or has its 3x3 rotation matrix altered in some fasion), the resulting rotation matrix *is not normalized*. For the scope of this project normalization of the column vectors is unnecessary and a waste of calculations.

These optimizations allow a rotation or a translation operation in approximately four milliseconds.

3D Library Use for Movement

Lukesky's Reference Frame

Lukesky's movement mechanics restrict movement to a 2D plane. The 3D library can still be utilized to track movement in a plane. One frame was assigned to represent Lukesky's position and orientation in 3D space.

- The frame is positioned directly in the center of Lukesky.
- The positive Z-axis, Z^+, is oriented pointing forward; the negative Z-axis, Z^-, is oriented pointing backward.
- The positive X-axis, X^+, is oriented pointing to Lukesky's left; the negative X-axis, X^-, is oriented pointing to the right.
- The positive Y-axis, Y^+, is oriented pointing above Lukesky; the negative Y-axis, Y^-, is oriented pointing below Lukesky, directly at the ground.

The reference frame can be seen overlaid on Lukesky in the Figure below.

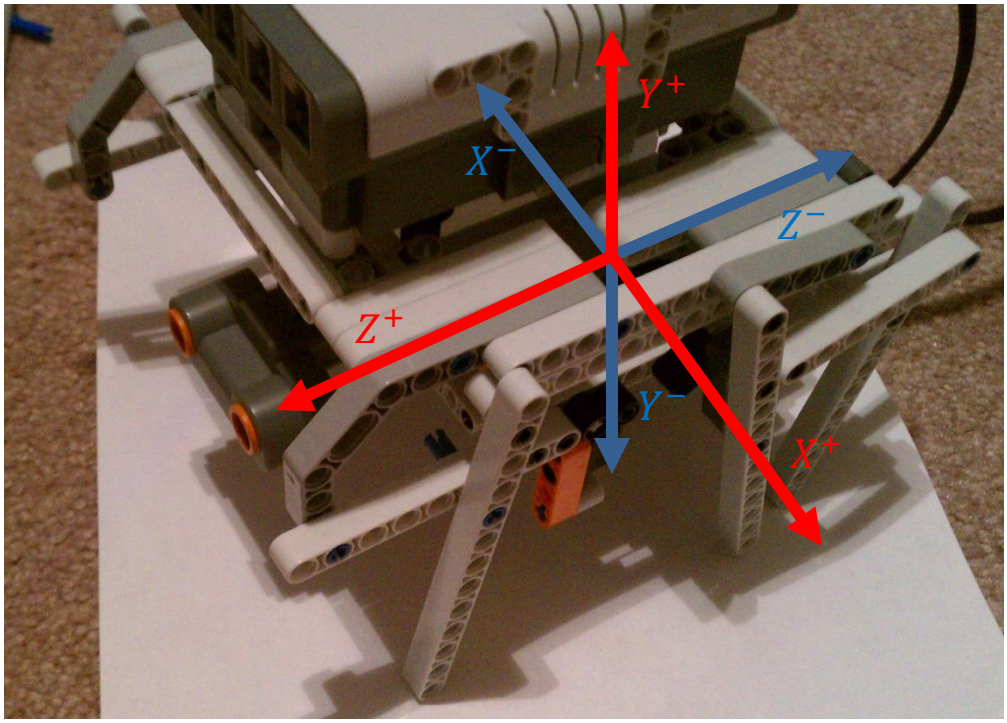

Figure 3 - The orientation of Lukesky's reference frame.

It can be seen that movement takes place in the plane defined by both the X and Z axes and that locomotion takes place along the Z-axis. Turns in the XZ plane are accomplished by rotating about the Y-axis.

Movement between Two Arbitrary Points

The 3D Library allows for creation and manipulation of any number of reference frames so long as there is enough memory to hold the frames. Using this feature of the library, a `Move()` function was created for Lukesky. The philosophy is simple: given two reference frames, *source* and *destination*, calculate the rotation and path necessary to move from *source* to *destination*. Since the `Move()` function is expected to be the only source of position manipulation for Lukesky, all of the path calculations execute when the function is called. An example:

```
_3D_translate_rel( currLoc, -10.0, 0.0, -10.0, nextLoc );
Move( currLoc, nextLoc );
currLoc = nextLoc;
```

The `_3D_translate_rel()` call translates the `currLoc` reference frame by `-10.0` units along both of its X and Z axes, then stores the resulting translated frame in `nextLoc`. This relative translation is *only for the reference* frames. This does not produce movement for Lukesky. The `Move()` call once passed the two reference frames with *source*=`currLoc` and *destination*=`nextLoc` then produces movement for Lukesky.

Move() Logic

The `Move()` function uses a specially made algorithm for Lukesky. The pseudocode for the `Move()` logic is shown below:

```
Move( A, B )
      Calculate current heading off of Z+ axis in A
      Calculate destination heading from point A to point B
      Calculate rotation direction from current heading toward B
      Calculate distance from A to B

      Rotate from current heading in A toward B
      Walk the distance from A to B
```

As can be seen in the pseudocode, each `Move()` consists of a rotation from the current heading toward the destination location, then a translation along the vector from A's location to B's location.

Calculation of Current Heading

A reference frame's Z-axis vector is used to calculate the angle [-180,+180] degrees off of the world Z^+ axis. This calculation is shown below:

$$\text{Current Heading} = \text{atan2}(Z_x, Z_z)$$

Which will return a radian value $[-\pi, +\pi]$ and will be positive if Lukesky has rotated to his left or negative if Lukesky has rotated to his right. This equation differs from those supplied in lecture due to lack of normalization of axes stored in reference frames.

Calculation of Destination Heading

To calculate the angle from the *source* frame to the *destination* frame, the differences in the X and Z locations of the two frames are used:

$$\text{Destination Heading} = \text{atan2}(x_B - x_A, z_B - z_A)$$

Taking the differences in the X and Z coordinates of the two frames produces a vector representative of the hypotenuse of a triangle centered at the origin. Then a call to the `atan2()` function results in a return of a radian value $[-\pi, +\pi]$ and will be positive if B is "to the left" of the current heading or negative if B is "to the right" of the current heading.

Calculation of Rotation Direction

To minimize the amount of time Lukesky spends rotating toward the *destination* point, Lukesky performs another calculation. Instead of always simply rotating left or right from the current heading toward *destination*, Lukesky makes the decision to rotate the smaller of the two possible angles.

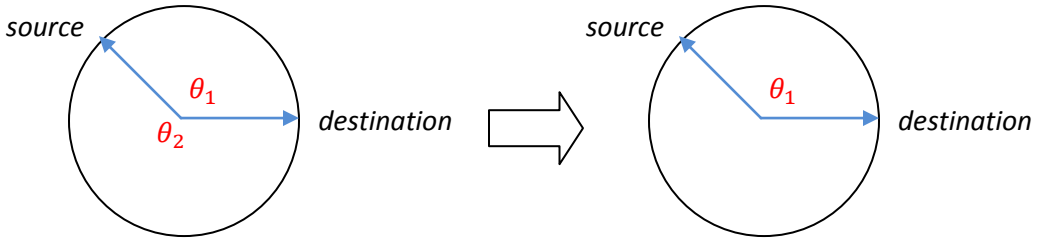

Figure 4 - A visual description of Lukesky's direction of rotation decision.

This is achieved by cleverly converting both the current heading and destination heading angles to "positive" values if either are negative. That is, if the current heading is -135 degrees, it becomes 225 degrees:

```
if current heading < 0
      convert to positive
if destination heading < 0
      convert to positive
```

Then if rotating left from *source* to *destination* would result in a rotation ≥180 degrees, rotation to the right happens, otherwise rotation to the left happens:

```
if current heading < destination heading
      if current heading + 180 degrees < destination heading
            rotate right instead of left

else if destination heading < current heading
      if destination heading + 180 degrees < current heading
            rotate right instead of left
```

Calculation of Distance from Current Position to Destination Position

The final piece of information necessary to move from A to B is to know the distance from A to B. This is calculated using a simple 3D distance formula on the (x,y,z) location vector of A and B:

$$\text{Distance}(A,B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2}$$

Since Lukesky's movement is restricted to the XZ-plane, the formula could reduce to:

$$\text{Distance}_{XZ}(A,B) = \sqrt{(x_A - x_B)^2 + (z_A - z_B)^2}$$

However this is not the case for this implementation. This implementation simply has y_A and y_B both equal to zero.

Translating Lukesky

Translating Lukesky is the same as `Move()` ing Lukesky to a *destination* reference frame which has only undergone a `_3D_translate()` call.

Rotating Lukesky

Rotating Lukesky is the same as `Move()` ing Lukesky to a *destination* reference frame which has only undergone a `_3D_rot()` call.

Rotations are executed by driving forward the leg opposite of the direction of the rotation. For a rightward rotation, the right set of legs is held at stop and the left legs are driven forward. The friction of the right legs in contact with the ground hold Lukesky's right side in place and force a rightward rotation:

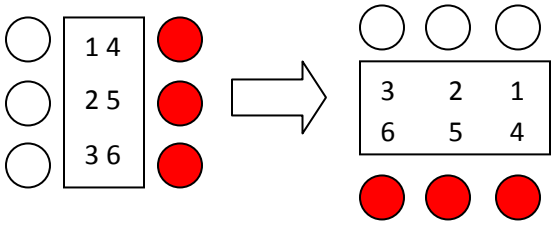

Figure 5 - A rightward rotation of 90 degrees.

A leftward rotation is analogous: the left legs are held at a stop and the right legs are driven forward.

Mapping Mathematic Movement to Real World Movement

To accurately move Lukesky about the real world with some form of coherence, it was necessary to map the motor rotations to a standard in the real world. Centimeters were arbitrarily chosen as the real world map value.

Translation

The `_3D_translate()` functions all expect parameters as a number of centimeters in the real world. Given the gear ratios of the output for the motors, the number of degrees the motor itself must rotate to produce one *step* for the output of the Gear System is:

$$\text{Degrees per Step} = \frac{Gear\ 2}{Gear\ 1} * 360 = \frac{36\ teeth}{12\ teeth} * 360 = 720\ degrees$$

To calculate the distance Lukesky covers per step, Lukesky was instructed to take *five* steps forward. This procedure was repeated six times. It was observed that Lukesky would travel farther when on a frictionless surface (ice or linoleum) versus a surface with friction (carpet) since his legs are plastic. It was then necessary to differentiate the centimeters per step traveled between the two types of surfaces:

$$\text{Centimeters per Step}_{Frictionless} = \frac{\frac{44.5 + 40.5 + 42.0 + 43.2 + 43.0 + 40.1}{6.0}}{5.0} \approx 8.44\ \frac{\text{cm}}{\text{step}}$$

$$\text{Centimeters per Step}_{Frictioned} \frac{\frac{31.0 + 35.2 + 33.7 + 33.9 + 36.3 + 32.0}{6.0}}{5.0} \approx 6.74\ \frac{\text{cm}}{\text{step}}$$

The requested real-world translation values were converted to motor rotation values using these centimeter per step values.

Rotation

The _3D_rot() functions all expect a real-world rotation value in degrees. To achieve real-world rotations of the desired degrees, it was necessary to calculate the number of degrees a motor needs to turn to complete on full *revolution* for Lukesky. Then for a `Move()` operation, a fraction of the full revolution would be rotated by a motor, resulting in the desired amount of real-world rotation. Again it was observed that slippage occurs on a frictionless surface, and a fix was introduced.

The *Bot Width* equals the width of Lukesky from one middle leg to the other and is approximately equal to 17 centimeters. The *Slip Fix* introduced adds extra rotation when turning to account for slippage of the feet.

$$\text{Rotation Circumference} = 2\pi * \text{Bot Width}$$

$$\text{Degrees per Revolution}_{Frictionless} = \text{Degrees per Step} * \left(\frac{\text{Rotation Circumference}}{\text{Centimeters Per Step}_{Frictionless}} + \text{Slip Fix}\right)$$

$$\text{Degrees per Revolution}_{Frictioned} = \text{Degrees per Step} * \frac{\text{Rotation Circumference}}{\text{Centimeters Per Step}_{Frictioned}}$$

Technical Challenges

During method implementation multiple unforeseen cases arose. The most notable ones are listed here with their applied solutions:

- The NXTAPOD had only forward motion and right-turn ability. Lukesky needed full range of motion.
    - Lukesky was equipped with the Twin-Motor system to allow both sets of legs to operate independently and allow rotation in addition to forward and backward motion.
- Lukesky would veer off course moving between points if the legs were not almost exactly one half motor rotation out of phase.
    - The NXC language allows for fairly precise control of the motors. A software piece controls the motor rotation, keeping the motors almost exactly in sync as long as the starting position of the motors is almost exactly in sync.
- The Lego Mindstorm NXT 2.0 motors are inaccurate and often overshoot or undershoot the desired degrees of rotation by large amounts.
    - The `Move()` function attempts to stop motor rotation as closely to the desired degree as possible. However, some error still exists. After the first, large rotation, the error in the rotation is calculated. The motors are rotated iteratively until the error falls under an acceptable amount.
- Use of the 3D reference frame equations for inverse kinematics doesn't work well for 2D planes of motion.
    - The inverse kinematics equations were re-derived specifically for this project.
- The Virtual Machine that runs the special OS capable of executing the NXC executable clocks input and output only once every one millisecond. The motor rotation counts and reset times are slow because of this.
    - The times are slow but inserting a `Wait( MS_1 )` after each motor rotation count reset or before each motor rotation count check gives the VM ample time to "catch up" to the code.
- There are many bugs with NXC
    - Arrays inside of struct types do not instantiate properly.
    - Arrays inside of struct types cause type mismatch errors when passed up or down a scope level more than one time.
    - Pass by reference does not exist, it is actually two pass by value statements.
        - These bugs were programmed around using a standard float[12] array instead of a struct called a _3DFrame containing a float[12] array.
- Lukesky moves differently on a frictionless surface than on a surface with friction.
    - Lukesky's movement capabilities were measured on surfaces with varying coefficients of kinetic friction. This has been accounted for in the conditional compilation of Lukesky's program.

Future Work

- Color sensing and tracking using the Color Sensor.
- Obstacle avoidance using the Ultrasonic Sensor.
- `RMove()` to have Lukesky walk backward from one point to another.
- `RStep()` to have Lukesky walk backward.

Acknowledgements, Sources, and References

Acknowledgements

The initial design of the NXTAPOD is by Daniele Benedettelli. [1]

Most of the optimizations for the NXC code come at the suggestion of John Hansen (bricxcc@comcast.net) the creator of NXC. His assistance in understanding and using NXC was invaluable during this project.

The initial idea for the directional choice when Lukesky rotates before moving to a point is credited to Rebecca Carrender.

Sources and References

1. http://robotics.benedettelli.com/NXTAPOD.htm
2. http://bricxcc.sourceforge.net/nbc/
3. http://bricxcc.sourceforge.net/
4. http://bricxcc.sourceforge.net/nbc/nxcdoc/nxcapi/index.html
5. This is a reference to the Star Wars universe, not of my own making.

Appendix

Pictures of Lukesky

In order to preserve as high of quality of pictures as possible and prevent downscaling, most pictures taken of Lukesky are not stored in this document. The pictures can be found in an SVN repository here:

http://code.google.com/p/my-random-cpp-libraries/source/browse/#svn%2Ftrunk%2Fprojects%2Funiversity%2Fcs345

Or they can be viewed online here:

https://docs.google.com/leaf?id=0B_s5yMVHMC16N2ZiZTYyOTYtNzYxNi00MDc0LTg3NTItMmEzNGNhM2ZkNzc2&hl=en

The building instructions can be found at reference [1] for the NXTAPOD, and the design can be altered from there if recreation of Lukesky is desired.

Source Code

The source code for this project is not stored in this document due to formatting annoyances. The source code can be found in an SVN repository here:

http://code.google.com/p/my-random-cpp-libraries/source/browse/#svn%2Ftrunk%2Fprojects%2Funiversity%2Fcs345

Contact

For questions or comments on this project, email Gary Steelman at gws4g2@mst.edu.