Patterns of Building LEGO® MINDSTORMS® Robots

KYLE BROWN, IBM Software Services for WebSphere

This paper discusses patterns found in the design and programming of robots built using LEGO® MINDSTORMS®, particularly those suitable for FIRST LEGO League (FLL) competitions. The paper is drawn from observations over five years of watching teams build LEGO Robots as an FLL coach, and three additional years as a competition judge.

Categories and Subject Descriptors: **K.3.2 [COMPUTERS AND EDUCATION]**: Computer and Information Science Education — *Computer science education*

General Terms: Robotics, LEGO, STEM Education

Additional Key Words and Phrases: None

ACM Reference Format:

Brown, K 2013. Patterns of Building LEGO® MINDSTORMS® Robots. PLoP 2013 Proceedings, 33 pages.

1. CONTEXT: INTRODUCTION TO FIRST® LEGO® LEAGUE

Interesting children, particularly middle-school age children, in science and technology and inspiring them to eventually pursue careers in STEM (Science, Technology Engineering and Mathematics) fields is a challenging and difficult task. However, one thing that all children share is a love of play, and most also find that they can perform their best in an environment that allows them to learn and grow as part of a team of their peers. One extremely useful mechanism that I have found for accomplishing the goal of creating an interest in science while teaching children skills that will be useful in STEM careers has been the FIRST® LEGO® League (hereafter FLL) program. From 2007-2011 I was a team coach for an FLL team, and then from 2010 through the present day, I have judged at FLL competitions. Through this pattern language I hope to capture and formalize my observations about what it takes to build a successful FLL robot and at the same time illustrate the applicability of the pattern form to this unique area of engineering.

FIRST® is a non-profit originally founded by inventor and entrepreneur Dean Kamen in 1989 to inspire young people's interest and participation in science and technology. According to their official web site (FIRST, 2013), "The mission of FIRST® and FIRST® LEGO® League is to inspire young people to be science and technology leaders, by engaging them in exciting mentor-based programs that build science, engineering and technology skills, that inspire innovation, and that foster well-rounded life capabilities including self-confidence, communication and leadership."

The FLL program works in the following way: Children are encouraged to form teams, working with adult mentors, of between 2 and 10 of their peers, between the ages of 9-14. These teams may be school-based, neighborhood-based, or organized through existing children's organizations such as 4-H, The Girl Scouts, or others. The program encourages problem solving and creativity by presenting kids with a real-world problem based on a new scientific theme, or *challenge* each year. The program consists of two parts; these are a technical presentation on the challenge theme, and what is called the *Robot Game*. The Robot Game is the subject of this paper.

2. MOTIVATION: THE FIRST® LEGO® LEAGUE ROBOT GAME AND LEGO® MINDSTORMS® ROBOTICS

The Robot Game involves the children in designing, building, programming, testing, and operating an autonomous robot that must perform a series of missions on a specially designed competition table. An example of such a table is shown on the following figure (Fig.1):

Fig. 1. FIRST LEGO League Competition Table

These robots are built using the commercial LEGO® MINDSTORMS® robotics building kits, and are programmed using the LEGO® NXT-G programming language. LEGO® MINDSTORMS® combines traditional LEGO® Technic parts with an intelligent microcomputer brick and intuitive drag-and-drop programming software (NXT-G[1]). NXT-G is a graphical language based on LabView from National Instruments (LabView). The robots must act autonomously while outside the small rectangular area in the lower right of this figure, known as "base". This means that the children have to not only design and build a robot of their own design that is mechanically capable of navigating around the board and operating the various mission models (built from LEGO® parts) to earn points, but they must also design, program and test the programs that instruct the robot how to solve these missions.

The culmination of the approximately 8-week FLL season is often in one or more tournaments, at which the teams compete with their peers in high-energy events that are reminiscent of sporting events, and also have a chance to present their solution to a real-world problem to a panel of presentation judges.

3. THE PATTERN LANGUAGE

This pattern language is written in a modified *Alexandrian* style, which is the style of the patterns that *A Pattern Language: Towns, Building, Construction* was written in. Given the physical nature of MINDSTORMS® robots, I believe that the Alexandrian style is an extremely appropriate one since Christopher Alexander's original work (Christopher Alexander, 1977) was also with physical objects, although at a vastly larger scale. What this means is that each pattern begins with a name (always in CAPITAL LETTERS) that has a number preceding it. The patterns in the pattern language are numbered from 1 to 14. Next there is a picture that illustrates the solution that the pattern is solving - this gives you a quick visual that helps you identify if this solution applies to your particular problem. Each pattern then has several sections of text that follow the picture.

The first section introduces the *context* of the particular problem that each pattern addresses. The context tells you when you may be able to apply this pattern. Then there is a divider consisting of three asterisks ("***"). Next, in **bold font**, there is a statement of the problem that the pattern is meant to solve. This

[1] For a tutorial of the proprietary NXT-G language, see (Yocum)

problem statement ends with the word "Therefore:", which is then followed by a statement of what the recommended solution to that problem will be. The solution is also highlighted in **bold font**. After the solution statement the pattern discusses what makes this program hard, or challenging. These are often called the *forces* that act on the problem. Another three asterisks divides the solution from the discussion of what happens when this solution is applied (which is sometimes called the *resulting context*). That discussion usually includes references to other patterns that may need to be applied as a result of applying the solution of this pattern.

This pattern language is intended to be used by the children involved in the Robot Game, and to a lesser extent, by their coaches. One of the FLL core values is that "The Kids do the Work". By targeting this pattern language directly toward the children, it can help them to complete the tasks they need to accomplish; their adult mentors and coaches can also use this pattern language to provide helpful suggestions to the children when necessary to help them overcome problems.

In this pattern language (which is a small subset of a larger language that has yet to be fully developed) I will examine patterns in the following areas, beginning with a set of simple foundational patterns about how the robot should be constructed, and then moving on to more complex patterns about attachment design and robot navigation:

3.1 Basic Robot Design and Strategy

This category of patterns includes the patterns that arise from the central problem of building a robot that can solve the robot game. The robot has to have certain engineering attributes like stability and modularity in order to successfully address this problem.

1. COLLECTION OF INTERACTING SUBSYSTEMS

3.2 Drive Base Design and Motile Appendages

The robot has to be able to move about the mat to succeed in the FLL challenge. These patterns address choices in this area.

2. DRIVE BASE
3. DIFFERENTIAL DRIVE
4. AUTOMOBILE DRIVE
5. WHEELED ROBOT
6. CASTER
7. TANK TREADS
8. SYNCHRO DRIVE

3.3 Attachment Design

This category of patterns come from the need to manipulate the various mission models that form the basis of the challenge. Different mission models necessitate different strategies for manipulation, resulting in different patterns.

9. PLOW ATTACHMENT
10. FORK TINES
11. LASSO ATTACHMENT

3.4 Navigation Mechanisms

The hardest problem in building an autonomous mobile robot lies in helping it sense its surroundings and navigate among obstacles to carry out its tasks. This category of patterns deal with those issues.

12. ODOMETRY
13. LINE FOLLOWING
14. WALL HUGGING

1. COLLECTION OF INTERACTING SUBSYSTEMS

When you are building LEGO® robots for an FLL competition, determining what your team's strategy for the year will be is the most important set of decisions the team must make. Chief among those strategic decisions is determining what their robot will look like and how it will function. However...

***

Many FLL teams do not perform at their best in a tournament because they do not have a coherent vision of how their robot is constructed.

FLL robots must accomplish several different tasks as described in that year's challenge documents; what's more these tasks change from year to year. However, as robot teams become more experienced through participating in more tournaments over the years, they find that there are certain types of tasks (such as moving an obstacle from one point on the board to another) that repeat over many different seasons.

What is common, though is that FLL teams view each year's robot as special-purpose implement for that year's challenge. Thus, the team will want to entirely "take apart" their robot at the end of the year. While this is a cathartic experience for the team, it also eliminates the accumulated engineering knowledge that the robot represents. But this is because the team views the robot as a specific object created for one purpose, and one purpose only - solving that year's challenge. Instead, the best FLL teams have learned to look at their robot in a different way. Therefore:

Build your robot as a collection of interacting subsystems, some of which will be reusable, and only some of which will be special purpose. Try to reuse the subsystems from year to year.

***

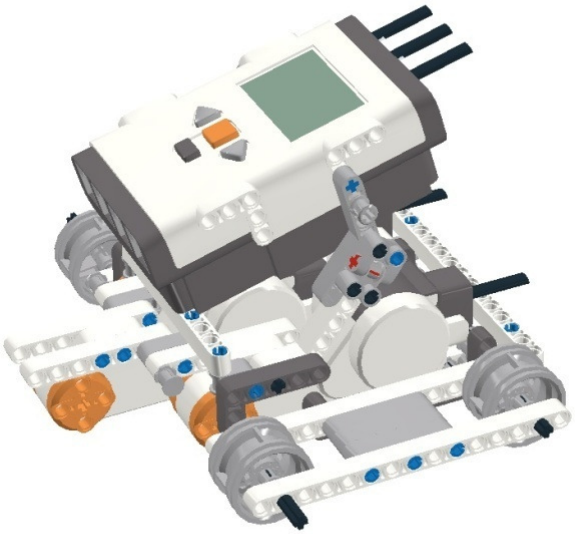
By thinking about your robot in this way, you can learn to evaluate what worked and did not work from each year's robot, and save the parts that did work, while removing and reworking the parts that did not work. In order to do this, think about your robot as a set of different subsystems for accomplishing different purposes. By far the most common subsystems are a DRIVE BASE (2) and a set of one or more attachments such as a PLOW ATTACHMENT (8) that solve various missions in the challenge.

An FLL team should realize that this principle is fundamental to complex system design in all areas. For instance, cars are divided into interacting subsystems as well. The engine of a car is made up of many parts, but it interacts with the transmission, which is also made up of many parts and with the steering system, which is also made up of lots of different component parts. However, when you call a mechanic to describe what's wrong, you use words that indicate to them which subsystem has an issue - such as "my engine won't start" or "the car makes a funny noise when I turn to the left". Likewise, designing a house consists of building many related subsystems, like the building frame, the electrical system, and the plumbing and water system. If you think about what subsystems your robot is made up out of, then you can also start to think about how to divide the work up among your team members along subsystem lines. Team members that understand this principle will become better designers of anything they will need to design and build throughout their lives.

Teams that are very advanced also discover that their programming tasks can be divided into subsystems as well. You can build reusable procedures to implement those subsystems using the NXT-G "MyBlocks" feature. These programming modules can also be reused from year to year by team members. For instance, a very common MyBlock that a team should build is a LINE FOLLOWING (11) MyBlock.

A potential pitfall of looking at a robot as a COLLECTION OF INTERACTING SUBSYSTEMS (1) is that you may miss potential synergies among subsystems if you view them too independently. For instance, you may not see the potential for reuse among subsystems if you follow this principle too closely and create very specialized subsystems to solve every mission model in the challenge.

2. DRIVE BASE



You are building a robot as a COLLECTION OF INTERACTING SUBSYSTEMS (1). Your robot needs to be able to solve many missions, but the Robot Game itself is of limited duration, so you can only make a limited number of changes to your robot within the time allotted for each round.

***

Your robot cannot function well unless it has a stable base that allows for both precise navigation and rapid attachment of mission manipulation attachments.

When you build a robot as a COLLECTION OF INTERACTING SUBSYSTEMS (1) you will need to be able to connect the different subsystems together. Your robot also needs to be able to move in certain ways - it needs to be able to move forward and backward, and also turn right and left. A robot that cannot turn, cannot move to all the parts of the board and thus can't solve all the missions.

You also need to allow for common tasks such as connecting your brick to a USB cable, and performing maintenance on your robot (for instance, changing disposable batteries or charging rechargeable batteries). Finally, you need to provide attachment points for your attachment that solve particular missions to connect onto, such as a PLOW ATTACHMENT (8) or FORK TINES (9).

Therefore:

Build a Drive base that moves the robot that is separate from the attachments that manipulate the mission models and other objects.

***

A drive base should consist of a sturdy attachment frame with the drive motors, gears, axles or any other part of the drive system and the controller brick attached to the frame. This combination forms the core of your robot. You can then add skids, casters or wheels to the frame to allow your robot to move.

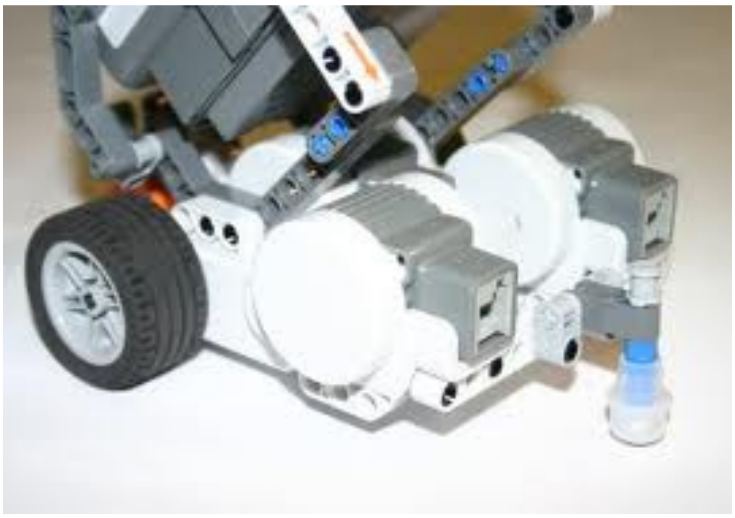
You do not want to attach your motors and other parts of your drive system directly to the NXT Brick because that makes it difficult to access the back of the brick to change batteries or even to attach the USB cable to the brick to program it. This also makes it difficult to change out the brick if necessary. So it's best if the Controller brick isn't a structural member of the drive base, even though it forms one in common designs such as the Tribot. The picture of a drive base shown above shows a better arrangement that allows for easy access to all the ports of the brick, and easily allows the controller to be removed for servicing.

When building your drive base, you need to make sure that your drive base includes holes in two different directions to allow for the placement of axles, as well as attachment of motors, wheels and other parts besides the NXT brick.

Given that mobility is a critical attribute for a robot, building a separate drive base has few drawbacks. However, your drive base will have to support the type of "motive appendage" that you choose to use to move your robot around the board. A "motive appendage" is simply the mechanism that you use to translate motion from the motors into motion of the robot. As an example, your legs are the motive appendages of your body. In the case of your robot, you may choose to build a WHEELED ROBOT (5), which may or may not need a CASTER (6), or you could choose to use TANK TREADS (7) attached to your drive base. You also need to decide on the arrangement of motors for steering left and right and moving forward and back. You will need to either use a DIFFERENTIAL DRIVE (3) or AUTOMOBILE DRIVE (4) for your drive base, or a SYNCHRO DRIVE (8).

However, one thing you may want to consider is the possibility of making one other subsystem a part of the drive base itself. For instance, a common strategy involves collecting several objects and bringing them into base. Having a holding pen as part of your drive base to hold these objects temporarily as opposed to making that a separate attachment may be a useful strategy in some cases

3. DIFFERENTIAL DRIVE



You are building a Robot as a COLLECTION OF INTERACTING SUBSYSTEMS (1). You are deciding how to build your DRIVE BASE (2) to allow for an arrangement of the motors that move your robot.

***

A drive base should use only a minimum number of motors, and should be both easy to program to and easy to build.

The NXT brick can itself only control three motors at a time, and the FLL rules only allow for a total of 4 motors to be used in a robot altogether. This means you want to leave at least one port open to control motorized attachments to manipulate mission models. What's more, younger teams find it difficult to build complicated drive bases, and programming more complex arrangements can be difficult as well.

Therefore:

Build a drive base founded on a Differential Drive approach that uses two motors, one on each side of the robot. As both motors turn in the same direction, the robot moves in that direction. As the motors turn in the opposite directions, the robot rotates in place around its center.

The figure below (Fig. 2) shows how a simple robot based on the differential drive approach can turn counter-clockwise. By reversing the directions of each wheel, the robot rotates clockwise.

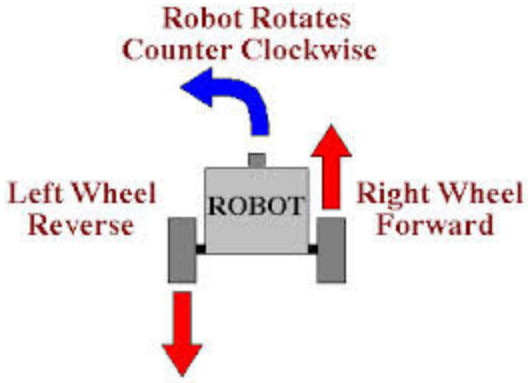


Fig. 2.: Differential Drive turning

A differential drive is by far the simplest type of drive system to build, and it is the one used by the Tribot robot whose building instructions are included with the MINDSTORMS® kit. This robot, which uses two motors placed side-by-side in a differential drive approach is depicted in the picture above. A simple differential drive will often have wheels directly attached to the motors, but more sophisticated ones will use gearing or other mechanisms for more flexibility in connecting the motors to the driving wheels.

What's more, a differential drive is easy to program to. The NXT-G "Move" block can simultaneously control two motors at a time, and can even send different power levels to each motor. With a differential drive, that means that a single Move block can tell a robot to go forward or back, or to move in a curve towards the left or the right. An example of a very simple program that moves the robot forward for 2 rotations, and then tells the robot to rotate in place around its center for 1/4 of a rotation of the wheel attached to the "B" port is shown below (Fig. 3). Even beginning team members can use this approach of ODOMETRY to make the robot perform complex navigation.

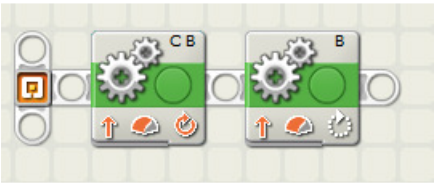


Fig. 3: Differential drive programming

A potential problem is that a robot built with a differential drive pivots around the center of the robot; making it difficult to position some attachments accurately in some cases. In those cases a SYNCHRO DRIVE (6) may be more appropriate.

4. AUTOMOBILE DRIVE

You are building a DRIVE BASE (2). Your robot does not need a tight turning radius.

***

Drive systems that travel over small obstacles or that drive longer distances (such as the entire length of the mission mat) may need very high torque or high speed. However, you still want to use a minimal number of motors

The problem with a differential drive is that the two motors serve the purpose of both turning and powering your robot. That means you can't make different choices about gearing for both driving and turning. You may need this, for instance, if your robot is quite heavy and thus you want to built your DRIVE SYSTEM (2) for high torque in moving forward or back. You do not need a tight turning radius

Build your DRIVE BASE (2) around an Automobile drive. This includes a powered axle in the back and a turning axle in the front.

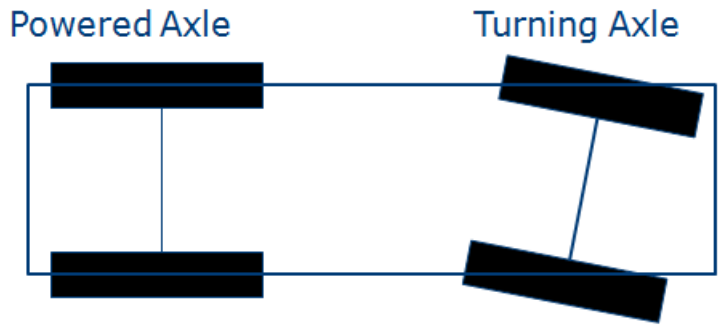


Fig. 4: Automobile drive aspects

The basic parts of an Automobile drive are shown in Fig 4 above. An Automobile drive needs only two motors - one in the front that turns the turning axle to the left or the right, and a separate one in the back that powers the fixed powered axle. One disadvantage of an Automobile drive is that the steering itself is often complicated to build. This is because the turning axle needs to be able to turn freely around its center axis but still remain attached to the robot. In many cases this means you will use a *rack and pinion* to turn

the front wheels. A rack is essentially a gear that has been flattened. A pinion is a small gear that turns on the rack and moves the rack back and forth. This principle is shown in the diagram below (Fig. 5).

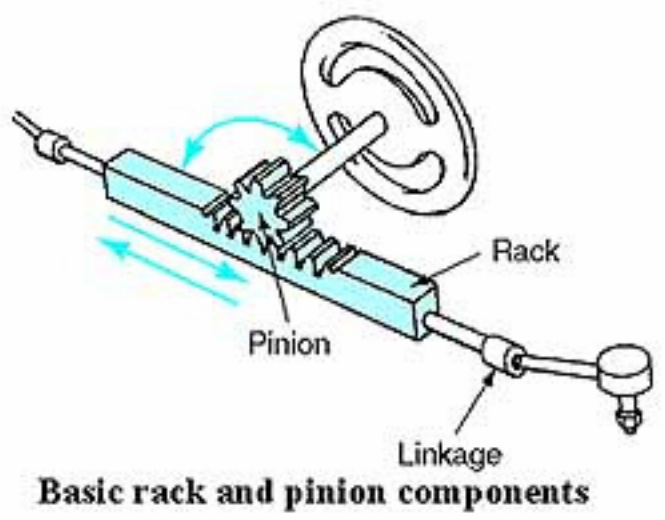


Fig. 5: Rack and Pinion components.

An example of a suitable rack and pinion steering mechanism built with LEGO is shown below (Fig. 6). Here, a motor to control the steering would be attached to the axle at the top of the model.

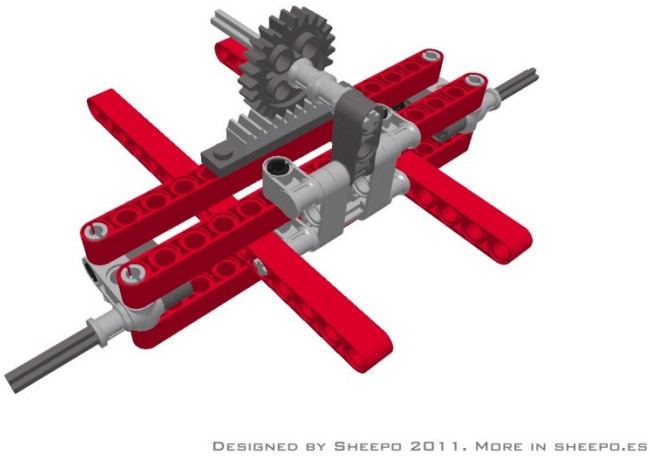


Fig. 6: Rack and Pinion steering made from LEGO parts.

A simpler option than a Rack and pinion is a 4-beam linkage system that uses LEGO three-hole beams with four snaps to simultaneously move two of the four beams at an angle. An example of this is shown below. The advantage of this approach is that a single motor with can be plugged into the snaps in the center directly and thus control the side-to-side movement. That is the solution that is used in the robot shown in the first picture in this pattern. A closer view of the construction of this linkage is shown in Fig. 7.

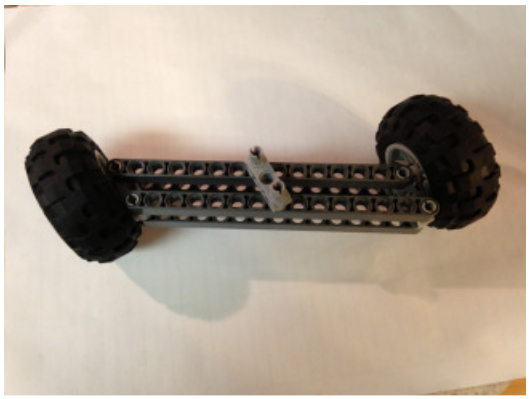


Fig. 7: Linkage made with 3-hole beams with 4 snaps

Since in an automobile drive, the steering wheels turn independently of the drive wheels, you can apply any amount of gearing for power or torque you want to the drive wheels. However, a ramification of this design is that the turning radius of the robot as a whole is not good - it is impossible, for instance, to rotate the robot around an axis as you can with a DIFFERENTIAL DRIVE (3). Another ramification is that since the turning axis moves, it makes it slightly more challenging to attach sensors or attachments to the front of the robot.

***

The turning axle should probably use wheels, making this a WHEELED ROBOT (5). However, for the power axle you can use either wheels or TANK TREADS (7). If you use TANK TREADS (7), then this kind of mixed arrangement is called a half-track, such as this French example from the 1930's (Fig. 8).

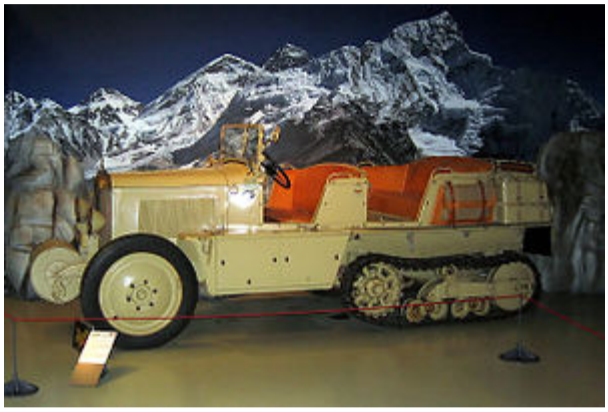


Fig. 8: Citroen Half-track from the 1930's

5. WHEELED ROBOT

You are building a robot as a COLLECTION OF INTERACTING SUBSYSTEMS (1). You have constructed a DRIVE BASE (2).

***

Motor rotation must be translated to movement on the board in order for your robot to solve missions. You only have a limited set of LEGO parts available, and you don't want to make things more complicated than they need to be.

FLL teams do not have an infinite amount of resources, and that certainly means that they don't have access to an infinite amount of LEGO parts. While there are many sources (e.g. LEGO Education, LEGO Stores, etc.) for specialized LEGO parts, many teams can only afford the parts that come with the base LEGO MINDSTORMS® NXT 2.0 kit. The kit does, however, come with a set of small tires and rims as well as a good selection of LEGO axles. It is relatively cheap and easy to find and purchase additional tires and rims in somewhat larger sizes as well.

Therefore:

Take advantage of the parts that you do have available in the base MINDSTORMS® NXT 2.0 kit and build a robot where the motors in your DRIVE BASE attach to wheels to move the robot around the board.

Probably the most common cases are two-wheeled robots (such as the Tribot) and four-wheeled robots as shown in the picture above. However, there is nothing that keeps a team from being creative and using three wheels, six wheels, or any other number. Another choice the team must make is in what size of wheels to use. Larger wheels allow the robot to move faster, but this may come at the expense of accuracy in navigation. With smaller wheels, however, the robot may be too close to the mat and attachments may drag on the mat and interfere with navigation as well.

When you consider attaching the wheels to the motors, there are a few options you can consider:

(1) You can directly attach the wheels to the motors
(2) You can use gearing and axles to change the rotation speed and torque of the wheels and make that different than the rotation speed and torque of the motors.

The first option is simpler, but in that case you are limited to the speed and torque of the motors. In the second case you can gear up for more speed than the motors can provide but at the expense of torque. In the section option, you can add more torque (for instance, for helping a heavy robot climb over an obstacle) but at the expense of more complexity.

***

One mechanical issue in building a two-wheeled robot with a DIFFERENTIAL DRIVE (3) is that a robot cannot stand on two wheels alone. For stability it needs at least one other point of contact with the mat. This can be a simple skid, additional wheels or a CASTER (6).

6. CASTER

You are building a robot with a DIFFERENTIAL DRIVE (3). You have chosen to use two wheels as your motive appendages as part of building a WHEELED ROBOT (4).

***

A two-wheeled robot cannot stand upright alone. It needs a third point on which it can meet the board. This third point needs to be able to allow easy movement in any direction.

A simple approach for providing the third point for a differential drive robot might be a simple skid, which is just a single LEGO® part such as a LEGO® Technic 45 degree angle beam that drags along the ground. That drag, however, produces unpredictable amounts of friction which can make the robot both slow and make precision navigation difficult. Another potential solution is to use a second, fixed set of two wheels attached to a single axle, but the problem is that while that allows for easy movement in the forward and backward direction, it makes moving right and left difficult as the tires drag along instead of turning with the robot.

Therefore:

Build a caster that allows easy motion in its current direction, but facilitates turning in place.

A *caster* is a common term for a mechanical object that you've probably seen before without necessarily knowing what it was called. For instance, wheeled office chairs are built on top of casters. The picture above shows just such a caster for use in office chairs. Casters are also used in shopping carts, and on many different types of portable furniture such as wheeled carts for audio-visual equipment. Casters can be built in several different ways, but the most common are to build them around wheels or balls. The key to building a successful wheel caster is that both the wheel axle and the center axle need to swivel freely. The picture below (Fig. 9) shows a LEGO® wheel caster with independent wheel axles and caster rotation axles.

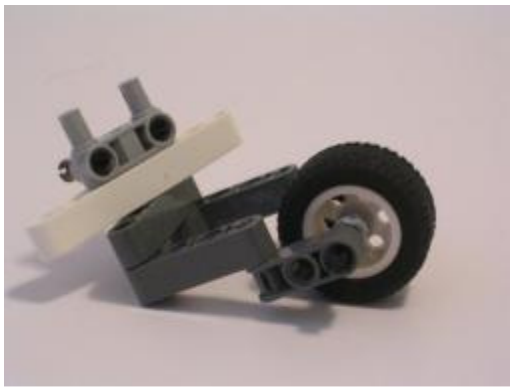


Fig. 9: Wheel caster built with LEGO® parts.

The problem with building a successful ball caster (which does not need axles) is in building a cage that can house the ball fully yet still allow for it to both rotate and be connected to the robot. One good solution for that using LEGO® 45 degree Technic beams is shown below (Fig. 10).

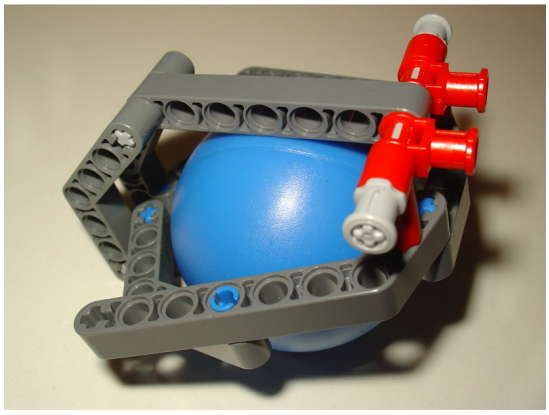
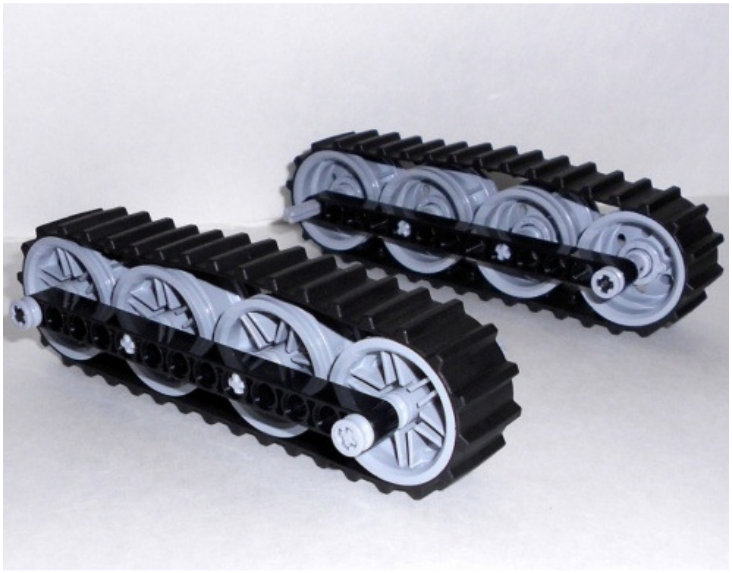


Fig. 10: Caster built with LEGO® Technic beams

A ball caster like that shown above allows free rotation in all directions just as easy. The problem with a stick caster is that it allows very easy rotation in the forward and backward rotation, but shifting to the left or the right may introduce a little bit of drag until the caster rotates on the center axle - this can make turns a little imprecise and not completely repeatable. As a result, even though ball casters are harder to build, they result in more accurate robots.

7. TANK TREADS



You are building a Robot with a DIFFERENTIAL DRIVE (3). In many challenges, there is an obstacle that must be climbed over, or onto, such as the bridge that was featured in the 2012 Senior Solutions Challenge shown below (Fig. 11).

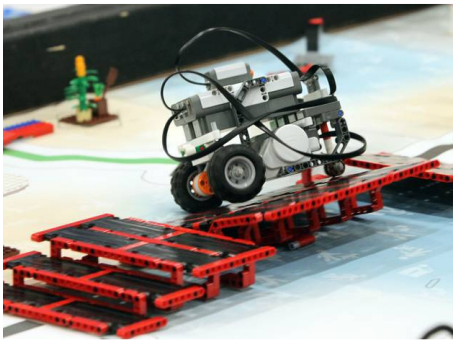


Figure 11: Robot on Bridge

***

Robots sometimes must climb over obstacles to solve missions in the most direct way possible. Wheels can make a robot unstable when it climbs over obstacles.

As you can see from the picture above (Fig. 11) a problem with a WHEELED ROBOT (4) is that each wheel only touches the mat (or an obstacle that it is climbing) on a very small part of the wheel. That means that if the wheel lifts up off the mat even just a little bit as it is climbing over an obstacle, that the wheel will spin in place and not get any traction. At best, this means that if the other wheel is still moving, that the robot will rotate uncontrollably, but at worst, it means that the robot can get "hung up" on the obstacle and the team will have to incur a "touch penalty" by retrieving the robot, thus both lowering their score and losing time from the round.

So even though a WHEELED ROBOT (4) is easy to build, there are certain times, such as climbing obstacles, when a WHEELED ROBOT (4) may not be the best approach.

Therefore:

Use tank treads as your motive appendages whenever you need to climb obstacles.

Instead of wheels and casters, use tank treads to gain traction. The Advantages of tank treads instead of wheels are that they provide good grip, especially when your mission strategy requires you to climb over obstacles such as the bridge shown above. This is because the tank tread contacts the mat over its entire length - even when climbing over an obstacle, some part of the tank tread will still be in contact with the mat or the obstacle and can provide traction. Likewise, even on slippery surface, tank treads provide relatively low slippage. An example of LEGO® tank treads is shown in the top picture above.

In the picture above you can see how you would power the tank treads with a DIFFERENTIAL DRIVE (3). The axle protruding from the wheel in front of each tread would be directly connected to a motor. In this approach, each motor provides power to a single wheel that drives the tank tread on that side of the robot. In mechanical terminology the powered wheel is often called a *driver wheel*. The remaining wheels along the tank tread are not connected to motors, and simply rotate along with the tank tread - they are called *follower wheels*. The simplest arrangement is for the driver wheel and follower wheels to be the same size, but that is not required - you can also arrange them either in a straight line (as the top picture shows) or in a different formation such as a triangle. An example of a triangular arrangement of tank treads and driver and follower wheels (although not built with LEGO®) is shown below (Fig. 12):

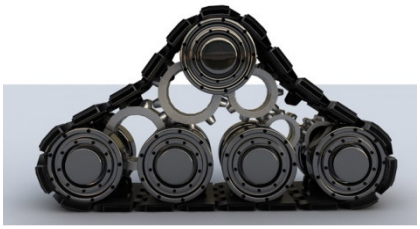


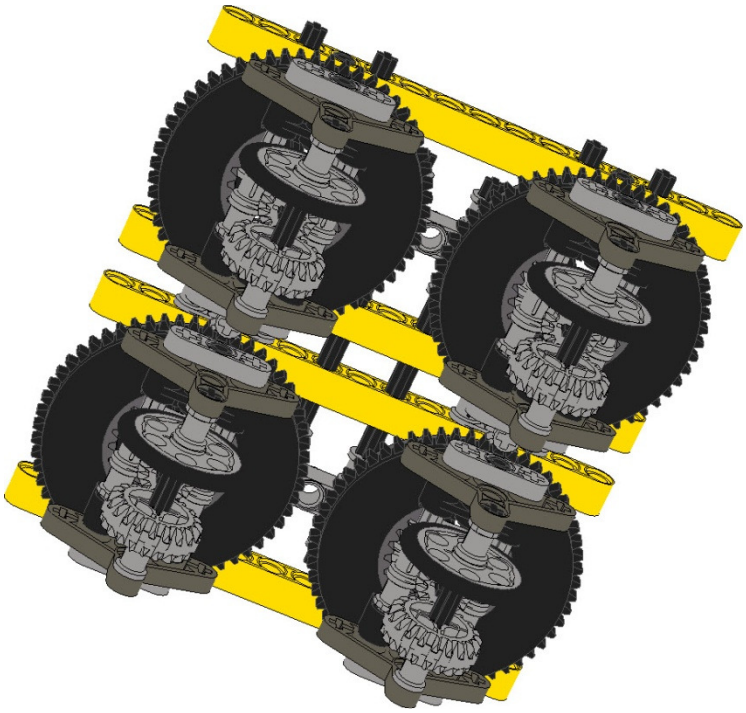
Fig 12. Triangular driver and follower wheels arrangement

***

The primary disadvantage of using tank treads (at least in many designs) is that the treads can move independently of the drive wheels that power the treads (e.g. the treads can slip). Tank treads can also sometimes have problems with driving in a straight line so any slippage on one side must be compensated for through navigation approaches or the robot will drift to one side or the other. A second disadvantage is that tank treads are not parts that come with the standard LEGO MINDSTORMS® 2.0 kit. They can be ordered online through LEGO Education however, and also are available as part of several commercial LEGO kits.

Because tank treads can have problems with slippage that cause unpredictable rotation, this can make accurate navigation with ODOMETRY (10) difficult.

8. SYNCHRO DRIVE



You are building a robot that is a COLLECTION OF INTERACTING SUBSYSTEMS (1), including attachments that grab objects. Sometimes solving a mission in the most obvious way requires grabbing an object and holding it in a particular orientation relative to the rest of the board while moving the robot. You need to devise a DRIVE SYSTEM (2) that takes this principle into account.

***

You are designing a robot that must turn in place without rotation.

Designing a robot to turn in place without rotating the robot is difficult when you have to consider the limitations of the drive system mechanisms so far considered. So, if your mission strategy and attachment design precludes the use of a DIFFERENTIAL DRIVE (3) or an AUTOMOBILE DRIVE (4), you may need to consider a different approach.

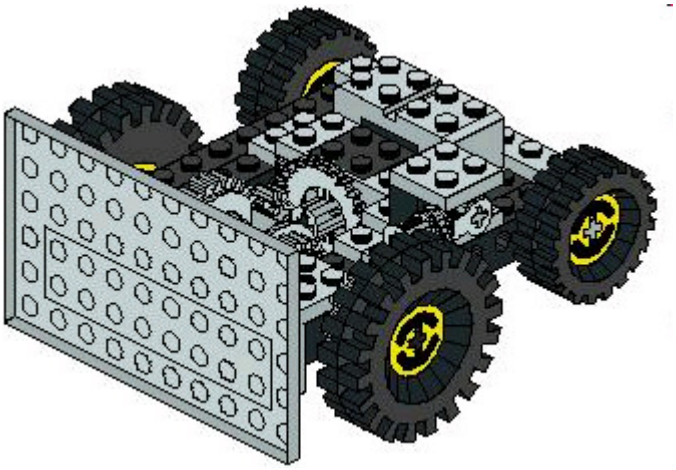
Therefore:

Build a robot with a Synchro drive

In a Synchro drive, all the wheels are simultaneously powered and turned. One motor provides the motive power to the wheels to move the robot forward, while another motor turns the wheels, using a LEGO® turntable to independently turn the wheels. An example of this is shown in the picture above.

The primary advantage of a Synchro drive is that it is extremely accurate and can, as noted, turn in place while holding an object stationary. Its primary disadvantage is that it is very complicated to build and is well beyond the abilities of most first or second year FLL teams with younger team members. I would recommend that only very advanced teams attempt building Synchro drives. However, if you want to try this on your own, then diagrams for several Synchro Drive designs exist, for instance (Squirrel, 2011).

9. PLOW ATTACHMENT



You are building a robot that is a COLLECTION OF INTERACTING SUBSYSTEMS (1), that has a common DRIVE SYSTEM (2)

***

One of your missions requires you to relocate an object from one location on the board to another.

Moving an item on the board can be complicated by the fact that robot navigation is not always precise. Let's say you try to try to move a standard LEGO brick across the board using an unmodified Tribot. The issue here is that you may be successful or unsuccessful in trying this depending upon what part of the front of the Tribot strikes the LEGO brick. The issue is that you need some way to gain some leeway in what part of robot strikes the target object, so that the effect will be the same regardless of the accuracy of your navigation.

Therefore:

Use a scoop or plow attachment to relocate the item

This is by far the simplest attachment to build, and usually the simplest to program for as well. In fact, problems that can be solved by a plow (or a slight variant that adds a wedge to the bottom forming a scoop) are the best ones to start teams on solving. The FLL mission designers usually intentionally add one or two missions that can be solved through this approach to each year's challenge in order to give new teams, especially of very young team members, a chance to feel a sense of accomplishment in solving something.

However, even with something as simple as a plow, there can be some disadvantages as well. One is friction - if you're not careful to lift your plow off the board a tiny bit then friction of the plow against the mat can interfere with navigation. Another is weight - adding a Plow attachment that is too heavy will make the robot front-heavy and may move the center of gravity on your robot too far forward, which can also interfere with navigation. To counteract that, you need to make sure your plow is as light as possible.

***

The missions best solved with a Plow usually take the form of requiring an object placed on the board in a specific location to be moved back to base. If the team can navigate their robot to a point behind the object, then they can move it with the plow to the base by moving the robot as a whole. This type of problem can be solved using ODOMETRY (10), and can be usually solved even with a simple robot built

using DIFFERENTIAL DRIVE (3) such as the Tribot that is featured in the instructions for the LEGO® MINDSTORMS® kit. Finally, a plow can sometimes improve the accuracy of LINE FOLLOWING (11) by acting as a light shield against extraneous light sources.

10. FORK TINES

You are building a robot that is a COLLECTION OF INTERACTING SUBSYSTEMS (1). Some common missions require specialization in their solution; for instance, one common mission model is the LEGO Loop, of which three are shown in the picture above. Loops are a common instance of an object that must be moved to base as part of a mission. However, the designers of the FLL challenge often make the problem more complex for the teams by placing these loops behind other objects, or on top of other objects, as opposed to simply placing them on the mission mat as shown above.

***

Your robot must grab an object with a hole that is placed a fixed height and deposit it in another location, potentially at a different height.

The problem with using existing solutions such as PLOW ATTACHMENTS (9) to relocate Loops or other objects is that many times the Loops are placed at differing heights on the board. A PLOW ATTACHMENT (9) can only work on items that are directly placed on the mat. You need a way to "thread" through the loop regardless of the actual height of the loop off the board.

Therefore:

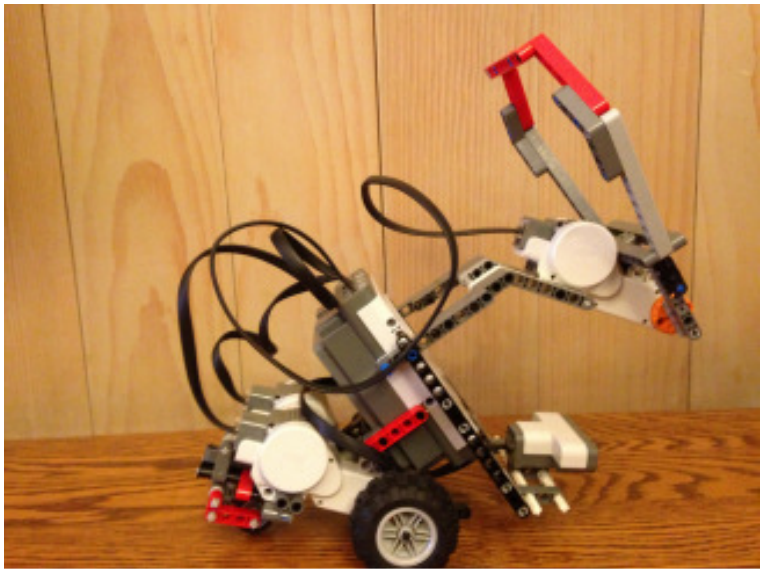
Build a simple fork tine attachment. The simplest version is a fork directly attached to a forward-facing motor. Multiple tines allow for variation in accuracy in getting to the loop. Single tines give more control but require precise navigation.

A fork tines attachment, which can move up or down, can usually address the requirements of this type of mission. An example of a powered fork tines attachment is shown in the picture above.

***

Fork Tines are easy to use with either ODOMETRY (10) or LINE FOLLOWING (11) navigation methods, but in the latter case, care must be taken with the positioning of the light sensor to not interfere with fork operation.

11. LASSO ATTACHMENT



You are building a robot that uses a DRIVE BASE (3). You need to solve a mission involving returning an object to base.

***

Objects that are positioned very close to another object are often impossible to move with a PLOW ATTACHMENT (9) or FORK TINES (10).

Many missions require your robot to grab an object and return that object to base. If the object is positioned close to a wall or close to another mission model then you cannot use a PLOW ATTACHMENT (9) to move the object because you can't position the robot behind the object to push it with the PLOW ATTACHMENT (9). If the object were a LEGO loop or other such object with a "hole", then you could use a FORK TINES to "snag" the object, but if it does not have a hole, then that approach will not work either.

What is needed is a way to in effect "throw a lasso" around the entire object and use the lasso to pull the object back to base regardless of where the lasso lands around the object.

Therefore:

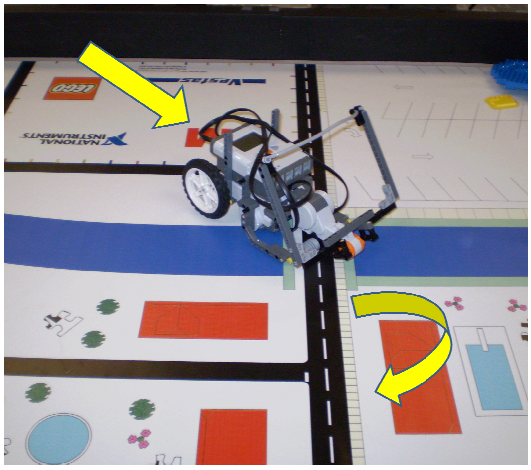
Build a Lasso Attachment that surrounds the object with a loop that is then pulled back. The lasso begins in the "up" position until the object is reached, when it is moved to the "down" position to surround the object.

***

Just as with the PLOW ATTACHMENT (9), a major potential problem with a lasso attachment is moving your center of gravity. in the model shown above, the weight of the lasso attachment and the ultrasonic sensor has already moved the center of gravity far to the front of the robot, lifting it up off its caster then the lasso is extended in the "down" position. Thus, you may want to add weight to the back of the robot to offset this and move the center of gravity back to the middle of the robot.

Given that the "down" position of the lasso surrounds the object and may also touch the mat, this can create drag that can interfere with navigation, especially if the team uses ODOMETRY (12) as their only navigation tactic.

12. ODOMETRY



You are building a robot , probably one with a DIFFERENTIAL DRIVE (3) such as the Tribot, but possibly one using other types of DRIVE SYSTEM (2). It needs to navigate to mission models, particularly those nearby to base.

***

Your robot needs to make simple turns and drive short distances with reasonable accuracy.

In order to solve many missions, especially those that just require simple attachments like PLOW ATTACHMENTS (9), you often only need to make the robot take a few simple turns and drive in straight lines to reach an object. When you are programming the robot to perform these tasks, you want to do so in a simple, easily understood way, especially one that can be comprehended by young children who are not familiar with complex programming techniques.

Therefore:

Use odometry as your method of navigation; measure your distances and turn radius and program the robot to move exactly that much.

Odometry is the approach of using information from sensors to determine travelled. Odometry is by far the simplest navigation method to program with NXT-G. What facilitates simple odometric programs is that the MINDSTORMS® motors have embedded rotation sensors built directly into the motor housings. NXT-G features a Move block that allows you to tell a motor to turn a certain number of rotations. For instance, the following program (Fig. 13) tells a robot with motors attached to outputs B and C to move forward 10 rotations, then turn to the right for 2 rotations of the left-hand wheel.

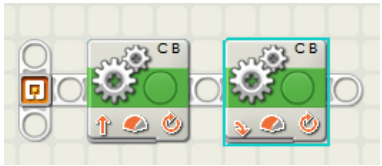


Fig.13.: Sample Odometric program

The problem is that odometric methods are not particularly accurate - they are very sensitive to the design of the robot, even requiring the matching of motors to remove slight differences in how fast they stop or how far they rotate. They are also particularly prone to wheel slippage and center of gravity variations.

***

Since a major problem with odometric navigation is accuracy, WALL HUGGING (12) can improve the accuracy in some cases. Where lines are available, LINE FOLLOWING (11) should be preferred since it has better accuracy.

13. LINE FOLLOWING

You are building a robot (probably using a DIFFERENTIAL DRIVE (3)) that needs to navigate precisely to a particular mission model. There are lines available on the mission mat near those models. For instance, the Food Factor challenge (shown above) featured many straight lines leading right to particular mission models.

***

In many cases, you need to precisely navigate to specific obstacles when a line is available leading directly to that obstacle.

ODOMETRY (12) has many drawbacks as pointed out earlier. Accuracy is chief among these drawbacks. However, often the designers of the challenges want to teach teams to use more complex navigation schemes by making navigation to an object difficult with ODOMETRY (12), but by placing straight lines very close to those mission models.

Therefore:

Navigate using a line following algorithm and a light sensor.

A light sensor is a LEGO part that contains a simple photocell that detects the intensity of light that shines on it. The LEGO part also contains an LED that provides light to the photocell. What the light sensor then measures when pointed at the mat is the amount of light reflected to it - which is an indication of the shade of whatever is directly under the light sensor. An example of the LEGO light sensor is shown below (Fig. 14):

Figure 14.: LEGO Light Sensor

A simple switch-based line following program is available in the NXT Education software instructions (LEGO Education). That program, written in the proprietary graphical NXT-G language, is shown below (Fig.15). It is appropriate for a simple robot with two driving wheels using DIFFERENTIAL DRIVE (3) such as the Tribot described earlier.

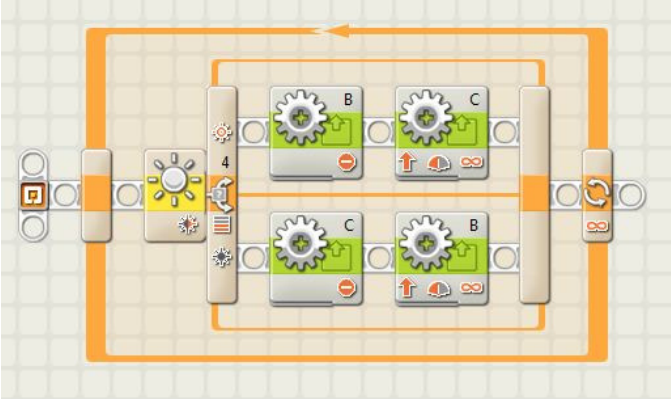


Fig.15.: Simple Line Following Program

Here we see an outermost loop containing an inner switch statement. The two conditions of the switch statements (the top and bottom inside the switch block), each contain two "motor" blocks that can control the activation or deactivation of a motor. In this simple program, which requires a light sensor to be attached to input port 4, the robot will turn to the left (the motor attached to port C will turn on and the motor attached to port B will turn off) if it senses light, and then turn to the right (the B motor will turn on and the C motor will turn off) if it senses darkness. Thus the robot will wiggle its way from left to right following the line - in this case forever, since the loop is an infinite one.

A difficulty that stops many team members lies is deciding how long to carry out that outermost loop. Looping for a particular count is not particularly accurate, as small variations on when you pick up the line will often vary the number of turns needed to complete following a line for a specific distance. Instead, adding another sensor (a light sensor or touch sensor) that looks for a separate condition in the outermost loop is usually the most appropriate approach. Another consideration that stops of confuses many team members lies in setting the appropriate thresholds for the values of light and dark. Small variations in the light sensor and in the printing of the mat need to be accounted for in setting these values, resulting in differences in how the robot will perform in testing and in competition. While the light sensors have a built-in light source, a light shield is also often needed to ensure that ambient light does not interfere with the light sensor measurements.

More advanced algorithms do more than comparing one value of light or dark - they have multi-way switch statements that compare multiple values and vary the amount by which the motors turn.

One very advanced PID (Proportional Integrative Derivative - a common algorithm used in machine control that integrates feedback) based line following algorithm can be found here: (Sluka, 2013) Others are available online as well for interested team members.

14. WALL HUGGING

You are building a robot that is using ODOMETRY (12) for navigation but you want to improve its accuracy. The robot needs to move a long distance down the board, particularly a long distance close to a wall.

***

You need to improve the accuracy of the robot over the long stretches when it moves a great distance down the board.

ODOMETRY (12) is inaccurate over long distances; many teams soon realize that while ODOMETRY (12) works over short distances, that over long distances it becomes very inaccurate - and worse, a slight variation on one turn or straight-away will make all the turns and straight-aways that follow go off in an increasingly wrong direction. However LINE FOLLOWING (13) is often considered too difficult for younger teams to master since it involves complex programming with loops and switch statements.

Therefore:

Use a wall-hugging approach that takes advantage of the fact that the FLL Competition table has a smooth wall a fixed distance from the edge of the mission mat.

A common technique is to mechanically modify the robot in a very slight way - that is to add one or (preferably) two LEGO® pulleys (as shown in the picture above) to one side of the robot, and then to introduce a very slight occasional turn into the Move block in the direction towards the wall. This will cause the robot to hug, or follow the wall on that side, thus increasing the accuracy in one dimension by fixing the distance of the robot parts to the wall. Especially when combined with a light sensor or touch sensor to find when to stop its forward motion, this is a simple, easy to program compromise for improving accuracy for odometric methods.

4. SUMMARY

In this paper we have described a number of different patterns that can help teams learn how to build successful robots for use in FIRST LEGO League competitions. In later papers we will examine additional patterns useful for building LEGO Robots for FLL competitions.

APPENDIX 1: PATTERN MAP

The following pattern map (Fig.16) illustrates the connections between the patterns. An arrow indicates that a pattern either is referenced by another pattern in its resulting context, or specifically references that pattern in its context. The patterns naturally fall into the five major categories of basic robot design and strategy, drive base design, motile appendages, mission attachments and navigation strategies as seen earlier in the pattern language.

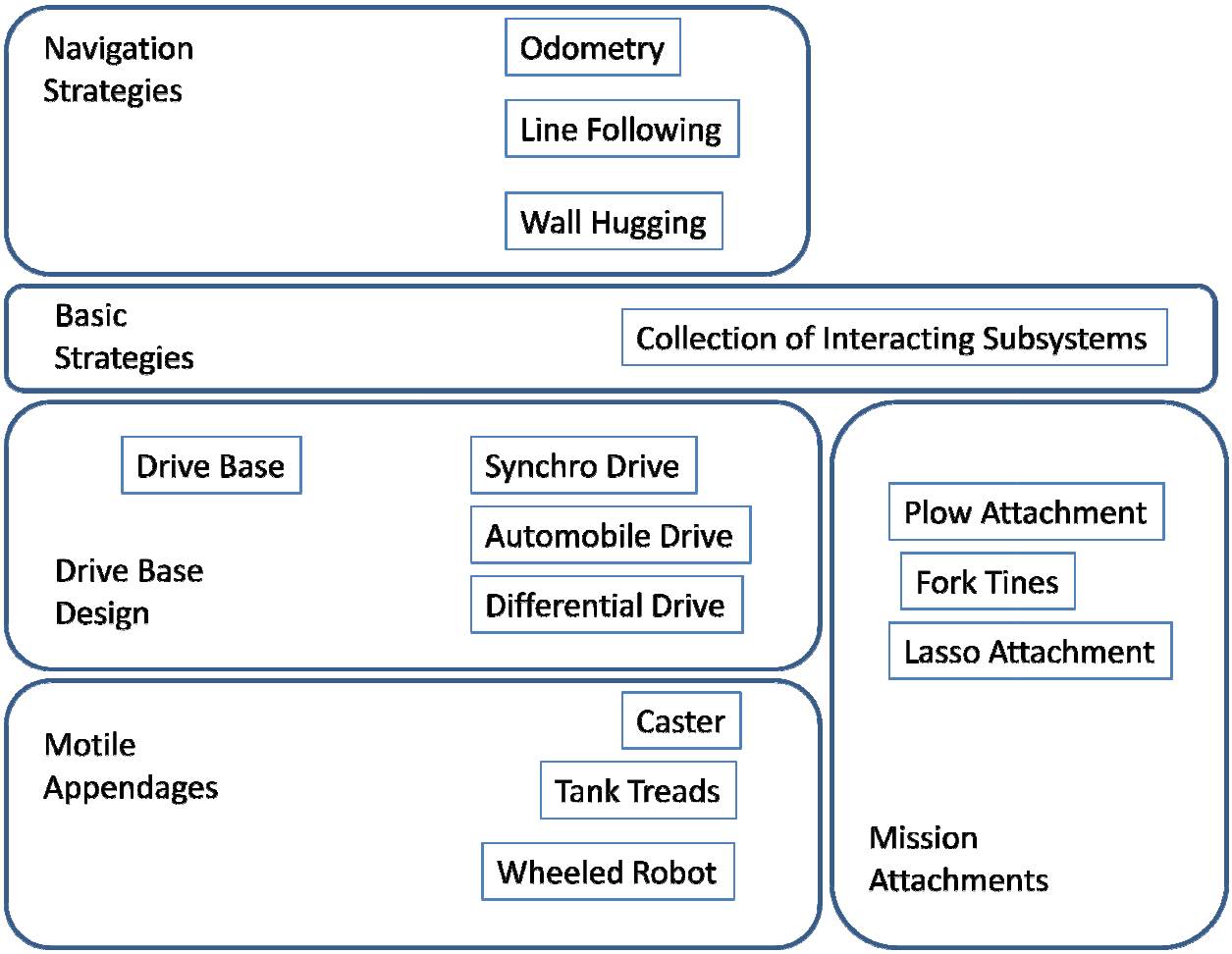


Fig.16.: Pattern Map

Appendix 2: Other Patterns

In this paper I have examined only a small subset of the full set of patterns that I have identified as being applicable to FLL robot design. Others that I will want to investigate in future papers include:

Robot Design and Strategy
Do the simplest thing that will possibly work
Mission Zones
Combined Zone Program
Mission start pushbutton

Attachment Design
Holding Pen
Spike
Pincer
Forklift
Dumper
Pusher Piston

Navigation Mechanisms
Offset Light Sensor
Light Shield
Color sensing
Obstacle detection

Next, there are additional detailed patterns of LEGO® design that can be helpful to team members sophisticated drive and attachment systems. These include:

Linear Motion
Rack and Pinion
Piston Rod
Lead Screw
Scissor Arm

Rotary Motion
Gear Train
Gear Ganging
Stop Bushings
Idler Gear
Bevel Gear
Crown Gear
Worm Gear
Clutch Gear
Ratchet
Pulleys and Belts

Finally, there are a few patterns about organizing your team, dividing the work and communicating between team members, coaches and judges. They include:

Team Organization Patterns
Subsystem Subteams
Engineering Notebook
Comment your Code

REFERENCES

Christopher Alexander, e. (1977). *A Pattern Language: Towns, Buildings, Construction*. New York, NY: Oxford University Press.

FIRST. (2013, June 17). *http://firstlegoleague.org/*. Retrieved from FIRST LEGO League.

LabView. (n.d.). *LabView Home page*. Retrieved July 29, 2013, from National Instruments: http://www.ni.com/labview/

LEGO Education. (n.d.). *LEGO NXT Education Software*. Retrieved July 29, 2013, from LEGO Education: http://education.lego.com/en-us/lego-education-product-database/MINDSTORMS®/2000080-lego-MINDSTORMS®-education-nxt-software-v-2-0/

Sluka, J. (2013, July 17). *PID Algorithm Description*. Retrieved from NXT Programming Website: http://www.inpharmix.com/jps/PID_Controller_For_Lego_MINDSTORMS®_Robots.html

Squirrel, B. T. (2011, December). *Synchro Drive in Lego*. Retrieved October 26, 2103, from Bob the Squirrel Blog: http://bobdasquirrel.blogspot.com/2011/12/synchro-drive-in-lego.html

Yocum, D. (n.d.). *StemCentric NXT-G Tutorial*. Retrieved July 29, 2013, from STEMCentric: http://www.stemcentric.com/nxt-tutorial/