4-2019

Integrating Mathematics and Educational Robotics: Simple Motion Planning

Ronald I. Greenberg
Loyola University Chicago, Rgreen@luc.edu

George K. Thiruvathukal
Loyola University Chicago, gkt@cs.luc.edu

Sara T. Greenberg
Loyola University Chicago, sgreenberg1@luc.edu

Recommended Citation

Ronald I. Greenberg, George K. Thiruvathukal, and Sara T. Greenberg. Integrating mathematics and educational robotics: Simple motion planning. To appear in Proceedings of the 10th International Conference on Robotics in Education (RiE 2019 in Vienna), Advances in Intelligent Systems and Computing. Springer-Verlag.

Integrating Mathematics and Educational Robotics: Simple Motion Planning

Ronald I. Greenberg*[1,2], George K. Thiruvathukal[1,3], and Sara T. Greenberg[1,4]

[1] Loyola University, Chicago, IL 60626, USA
[2] rig@cs.luc.edu [3] gkt@cs.luc.edu [4] sgreenberg1@luc.edu

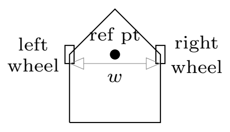
Abstract. This paper shows how students can be guided to integrate elementary mathematical analyses with motion planning for typical educational robots. Rather than using calculus as in comprehensive works on motion planning, we show students can achieve interesting results using just simple linear regression tools and trigonometric analyses. Experiments with one robotics platform show that use of these tools can lead to passable navigation through dead reckoning even if students have limited experience with use of sensors, programming, and mathematics.

Keywords: computer science education, robotics, mathematics, trigonometry, algebra, linear regression

1 Introduction

Providing robotics experiences has become a popular and successful mechanism for broadening participation in computing and STEM more generally, retaining more students in these fields, and improving their learning. For example, one study of student responses to brief computing outreach visits found that the most popular component of such visits was viewing of robotics videos [5]. The typical skill development focus in robotics programs is on logical thinking, computer programming, and/or engineering design. Mentors may be aware of the potential for integration with mathematics, but there is limited availability of level-appropriate materials focused on such integration for middle school and especially high school students. This paper focuses on motion planning built on the algebra and precalculus background typical for high school students.

We will work with a robot drawn as shown below that must navigate through a two-dimensional field. With two wheels that may be driven independently (forwards or backwards up to some maximum speed) and a third balance point such as a caster wheel or track ball, such a robot is generally referred to as a differential-drive robot, and this model is a good fit for most educational robots. Some of these platforms have a fixed distance w between the wheels, or *track*



* Supported in part by United States National Science Foundation grants CNS-1738691, CNS-1543217, and CNS-1542971.

width, but this parameter can be a point of flexibility in many LEGO®-based designs. Intuitively, a small w makes the robot able to move more nimbly, assuming one avoids values that are so small as to lead to problems with rollover.

We focus on dead reckoning through a known terrain without substantial sensor use. For experimental work, we do use a robot with a built-in functionality of testing the amount of rotation of the motors controlling the robot wheels. This makes the results more robust in the face of possibly varying levels of battery charge, but passable results might also be achievable using only timing delays as long as the battery is frequently recharged. Experienced teams in robotics competitions do tend to make use of other sensors that are typically available, for example a range finder to judge closeness of approach to a landmark or a reflectance sensor to detect black/white boundaries on the driving surface. While such sensors can provide still more robust results in the face of such phenomena as imperfections in the driving surface, we show here that students can use accessible mathematics to achieve passable results without advanced sensors and to compare different navigation primitives and strategies. In addition to providing an opportunity to integrate mathematical and statistical or data-driven reasoning, the results can be valuable to beginning and advanced teams. Beginning teams may not have yet mastered the use of sensors and the requisite programming, while advanced teams may desire a fail-safe in case of failure of a sensor, its mount, or its electrical connection. In all cases, dead reckoning is likely to be helpful, even if only for initial rough positioning before using sensors, since this often can be done at higher speed and lower power consumption.

Within this dead reckoning context, this paper uses an intermediate level of mathematics (algebra and trigonometry) to analyze and experiment with intuitive alternatives for basic navigation tasks. This contrasts with both very simple exercises suitable for students as young as elementary grades (e.g. [6]), and with studies using calculus or other heavy mathematics (e.g., inertial navigation, overviewed in [2, p. 77–80] for example, or proofs regarding optimal path types, using various constraints and criteria, for arbitrary changes in position and orientation of a differential-drive robot and even more complex car-like robots as in [1, 3, 7] and references therein). While Ben-Ari and Mondada [2] provide a rare example of explaining robotics concepts at primarily an intermediate level, this paper adds depth to their elementary discussions of odometry while stopping short of calculus-based discussions. (For a similarly intermediate-level discussion of integrating mathematics and data analysis into robot building, see [4].)

2 Navigation Framework and Primitives

We will think about moving a reference point on the robot from a start position at coordinates $(0, 0)$ to a target position (x, y) but keeping in mind that the movement is constrained by the use of our two wheels at separation w. We also assume the initial orientation, or heading, is $0°$, and the analyses in this paper relate to bringing the robot to a final orientation θ that is also $0°$; working with other ending orientations would not add a great deal of complexity. As another

```
/* doticks is called by other procedures to move wheels correct number of ticks */
void doticks(float leftlim,float rightlim){int lticks,rticks; /* to track left/right wheels */
  cmpc(LEFT); cmpc(RIGHT); /* cmpc & gmpc clear & get motor position counter */
  do {lticks = abs(gmpc(LEFT)); rticks = abs(gmpc(RIGHT));
      if (lticks>=leftlim) motor(LEFT,0); if (rticks>=rightlim) motor(RIGHT,0);
      } while (lticks<leftlim || rticks<rightlim);}
void gostraight(float dist){ /* dist in millimeters */ int ticklim = straightfit(dist);
  motor(LEFT,.9*speed); motor(RIGHT,speed); doticks(ticklim,ticklim);}
  /* .9 chosen empirically; straightens our robot using a primitive that sets motor speeds */
void rotate(float theta){ /* theta in positive/negative radians for right/left rotations */
  float ticklim; float dist=theta*w/2; /* rotation radius w/2 */
  if (dist>=0) {motor(LEFT,speed); motor(RIGHT,-speed); ticklim=rrotfit(dist);}
  else {motor(LEFT,-speed); motor(RIGHT,speed); ticklim=lrotfit(dist);}
  doticks (ticklim,ticklim);}
void swing(float theta){ /* theta in positive radians for right swings; negative for left */
  float dist=theta*w; /* swing radius w */
  if (dist>=0) {motor(LEFT,speed); motor(RIGHT,0); doticks(rswingfit(dist),0);}
  else {motor(LEFT,0); motor(RIGHT,speed); doticks(0,lswingfit(dist));}}
```

Fig. 1. Example code for straight, rotation, and swing movements. `LEFT` and `RIGHT` give port numbers controlling corresponding motors. `straightfit`, `rrotfit`, `lrotfit` `rswingfit`, and `lswingfit` incorporate linear regression results described in Section 5.

simplification, and to focus on the most common navigational tasks, we assume $y \geq w$, and we focus within this paper on $x \geq 0$, since negative values of x just lead to a mirror image. (Much of the complication of more advanced works is devoted to analyzing the much broader range of possibilities for x, y, and θ.)

We presume students will be most interested in comparing time from different navigation strategies rather than total amount of wheel rotation or some other criterion. To keep the math simple, we assume, as in some other works, that the robot can accelerate and decelerate instantaneously as long as a bound on velocity is respected. In practice, students may need to experimentally analyze effects of inertia and compose overall movements from a sequence of navigational primitives separated by brief delays that allow the robot to come to rest.

Advanced works regarding optimal trajectories of differential-drive robots show that optimal paths always contain only a limited number of segments of straight-line movements, rotations (about the reference point as one wheel moves forward and the other backward), and swings (with one wheel moving so that the robot pivots around the fixed wheel). These also are natural primitives for students to program.

To account for inertial effects, we recommend teaching students to gather data points experimentally and perform linear regression to obtain formulas for performing straight line motions of specified distance and rotations and swings of specified angle. While linear regression sounds a little advanced, it is actually an easy task using ubiquitous spreadsheet programs, but is still a meaningful way of integrating mathematical understanding into the motion planning task.

As an example, we gathered data for a LEGO-based robot built from parts provided in the Botball educational robotics program and wrote functions for straight, rotation, and swing movements parameterized by the distance to travel or the angle to turn through as in Figure 1.

Depending on the reliability of the routines to rotate or swing to a specified angle, students may also want to write routines that specifically turn 90° left and 90° right. In any case, we will begin by considering simplified navigation schemes in which all the straight-line motions are horizontal or vertical and the turns are all by 90°. Then we will consider more general navigation.

3 Horizontal and Vertical Navigation

With all navigation along horizontal and vertical lines with turns of 90°, the navigation primitives for turns can be simplified. Instead of gathering data and doing linear regression to write routines to swing and rotate through an arbitrary angle, one could just determine what is needed to turn right and left by 90°. In this context, an interesting first exercise for students (in addition to the testing and programming to create the navigation primitives) is to compare the effects on overall navigation from using rotations versus swings.

Figures 2(a) and 2(b) show the paths using rotations and swings when moving the reference point from $(0, 0)$ to (x, y) on a "middling" path through the terrain that is likely to avoid obstacles in a typical educational robotics setting. (If there is actually some more particular need to avoid an obstacle, the point where the first turn is taken can be easily adjusted without affecting the navigation time.)

Under the rotation approach of Fig. 2(a), both wheels are always in motion, so we can compute the time as being proportional to the distance traveled by either wheel, i.e., $x + y + \pi w/2$. Under the swing approach of Fig. 2(b), we can again consider either wheel, but when that wheel is stationary, we must account for the distance traveled by the other wheel; thus the time is proportional to $(x - w) + (y - w) + \pi w = x + y + (\pi - 2)w$. The swing approach is therefore superior at a time savings proportional to $2w - \pi w/2 \approx .43w$.

In practice, as previously noted, one may need a brief delay before and after each turn to allow for inertial effects where the theoretical path calls for a discontinuity in velocity, but we will have the same number of discontinuities, four (not counting start and end), whether using rotations or swings.

4 Generalized Navigation

While the navigational approach of the prior section is simple, we would expect to be able to navigate more quickly by proceeding on a path closer to a straight line. In addition, we can define paths with fewer points of velocity discontinuity at which we insert delays for inertial effects. As long as we have gathered enough data to calibrate the rotations and swings according to the angle desired, we just need to do some trigonometric calculations.

Figures 2(c) and 2(d) show the paths under the rotation and swing approaches when moving the reference point from $(0, 0)$ to (x, y) assuming we use the rotations or swings just to line us up for straight-line navigation. Both of these paths have just two (internal) points of velocity discontinuity.

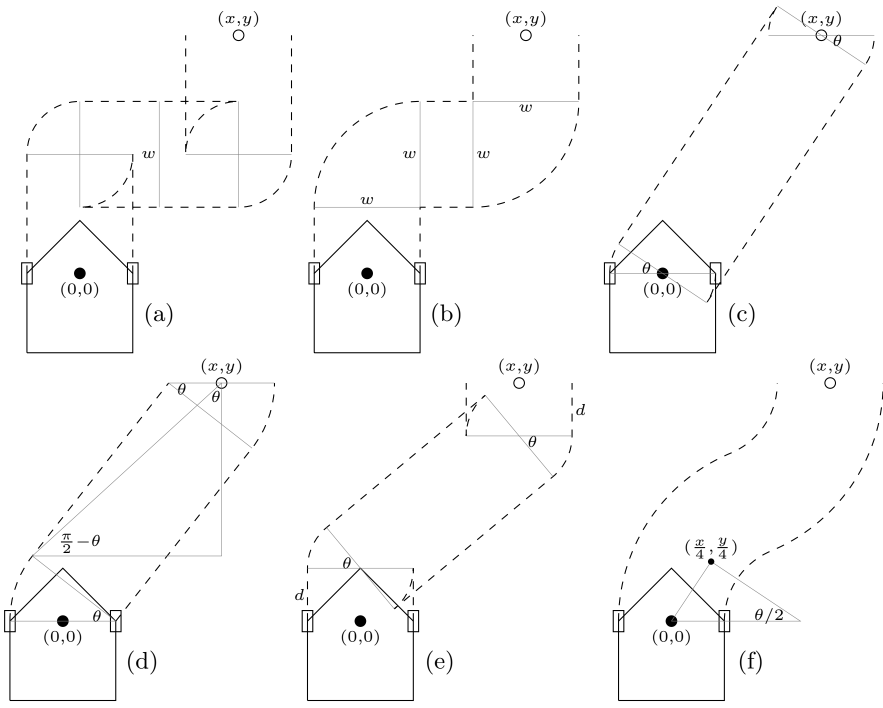


Fig. 2. The paths of the robot wheels for the routing methods considered in this paper. (a) and (b) for horizontal and vertical navigation, using rotations and swings, respectively. (c) and (d) for general navigation, using rotations and swings, respectively, and (e) and (f) for two other path types designed to avoid hitting a competition field boundary as may occur in (c). In (e), straight segments are added at the beginning and end of the path; these lengths are exaggerated for visual effect. In (f), we follow two mirror-image circular arcs.

Under the rotation approach of Fig. 2(c), both wheels are always in motion, so we can compute the time as being proportional to the distance traveled by either wheel, i.e., $\sqrt{x^2+y^2}+w\theta$ with θ in radians and $\tan\theta = x/y$. Under the swing approach of Fig. 2(d), we can again consider either wheel, but when that wheel is stationary, we must account for the distance traveled by the other wheel; thus the time is proportional to $\sqrt{(x-w+w\cos\theta)^2+(y-w\sin\theta)^2}+2w\theta$ with θ in radians and $\tan\theta = (x-w+w\cos\theta)/(y-w\sin\theta)$. We have not found an analytical solution for this θ, but writing $\tan\theta$ as $\sin\theta/\cos\theta$, cross-multiplying and using the identity $\sin^2\theta+\cos^2\theta = 1$, we can obtain a somewhat simplified function of θ for which we seek to find a zero: $y\sin\theta-(x-w)\cos\theta-w$. Students can be taught to write a simple routine using the bisection method to find a root in the range $[-\pi/2,\pi/2]$ Without an analytical solution for θ, it is difficult to make a full comparison of the times for rotations and swings, but we can at least say immediately that as x or y or both get large, the values of θ in these two cases approach the same value, and the swing approach takes longer by an amount of time proportional to θw.

With the observations just above, students can see that rotations seem to be better than swings for general navigation, in contrast to what was observed when using horizontal and vertical navigation. In addition, the math is simpler for rotations, but this navigational approach may often be impractical in robotics competitions, because the robot may often butt right against a boundary of the competition field at the start or end of the path and therefore be unable to do a pure rotation at that point.

There are two more simple navigation strategies that students might like to analyze to cope with the potential problem of boundary walls abutting the start and end positions as shown in Figure 2(e,f).

In Figure 2(e), straight segments of length d are added at the beginning and end of the path, while otherwise using the basic rotation approach. A sufficient value for d to avoid hitting a boundary running horizontally just below the robot would be the difference of the distance from the reference point to the lower right corner of the robot and the distance from the right wheel center to the lower right corner of the robot. The resulting angle of rotation is such that $\tan\theta = x/(y-2d)$, and the time spent on wheel rotation is $\sqrt{x^2 + (y-2d)^2} + 2d + w\theta$.

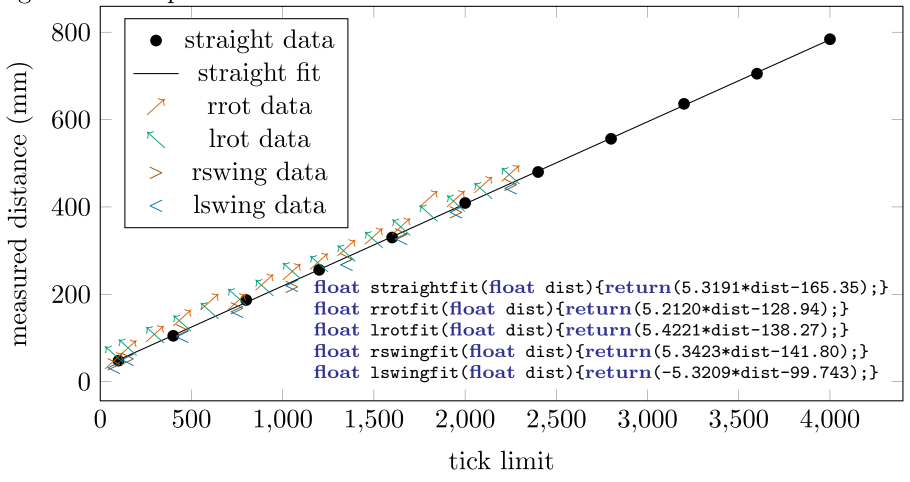
Typically, the displacement d should be very small compared to other distances so that it should have little effect on the angle and wheel rotation time. But another difference is that we are adding two more points of discontinuous velocity where we will insert delays to handle inertial effects; this brings us back to the same number of internal discontinuities, four, as when we restricted movements to be horizontal and vertical.

In Figure 2(f), we consider an approach that reduces the number of velocity discontinuities within the path to just 1. For $x \neq 0$, we route the reference point along a circular arc to $(\frac{x}{2}, \frac{y}{2})$ and then along a mirror image arc to the final destination. From the right triangle drawn for reference in the figure, we can see that the angle of rotation for each of the two arcs is given by $\theta/2 = \arctan(x/y)$, and the radius of the arc traced by the reference point is $r = (x^2+y^2)/(4x)$. The left wheel initially rotates on an arc of radius $r + w/2$ and the right wheel on an arc of radius $r - w/2$, and then the roles reverse. These arcs can be realized by running the outer wheel at full power and the inner wheel at a relative power of $(r - w/2)/(r + w/2)$; in practice, an adjustment may be needed for wheels that do not behave entirely identically, similar to the adjustment in relative powers that may be needed just to make the robot drive straight. Aside from such an adjustment, the total time spent on wheel rotation is proportional to the lengths of the two wider arcs, which is $(r + w/2)2\theta = ([x^2 + y^2]/x + 2w)\arctan(x/y)$. For x of large magnitude, the path traced by the wheels is longer than under previous methods but with just one internal delay for switching wheel powers.

5 Empirical Results

To perform empirical tests, we began by writing routines as in Figure 1 for straight, rotation, and swing movements parameterized by the distance to travel or the angle to turn through; these routines loop until reaching a specified value from a built-in gauge measuring the amount of rotation (number of ticks) of the motors controlling the wheel(s). We initially coded `straightfit`, `rswingfit`, `lswingfit`, `rrotfit`, and `lrotfit` to just return the argument given, and we gathered data for millimeters traveled or degrees turned when running the routines of Figure 1 with various limits. For turns, we separately handled positive (right turn) and negative (left turn) arguments. Then we fitted a line to each data set using a spreadsheet program; we also verified with a short R program. After fitting the lines, a simple algebraic manipulation allowed us to rewrite the corresponding navigation primitives to actually travel a specified number of millimeters or turn a specified number of radians. Students can exercise their trigonometry and algebra knowledge by solving for *ticklim* after replacing x with *ticklim* in the regression line from `gostraight` ($y = .188x+31.086$), $ticklim/w$ for `swing`, and $ticklim/(w/2)$ for `rotate`. They also must replace y in the regression lines by *dist* for `gostraight` and by $\theta\frac{360}{2\pi}$ for `swing` and `rotate` for radian/degree conversion. Finally, for `swing` and `rotate`, we also replace θ by $dist/w$ or $dist/(w/2)$, respectively. In all cases, we complete simple algebraic manipulations to express *ticklim* in terms of *distance*. The numerical results obtained appear in the code squeezed into the white space in the graph below.

It is hard to show all the data from our experiments compactly, but if we perform the above transformations to convert measured turning angles to distances (with $w = 150$mm for our robot), and look at absolute values of tick limits, then the data all falls very close to the regression line for the `gostraight` data as shown here, and we added code as shown based on the regression results and algebraic manipulations:



```
float straightfit(float dist){return(5.3191*dist-165.35);}
float rrotfit(float dist){return(5.2120*dist-128.94);}
float lrotfit(float dist){return(5.4221*dist-138.27);}
float rswingfit(float dist){return(5.3423*dist-141.80);}
float lswingfit(float dist){return(-5.3209*dist-99.743);}
```

Then we ran some tests of navigating along paths as defined in Figures 2(a–d) using general procedures we wrote, incorporating delays as short as 100ms at velocity discontinuities. We tested with (x, y) as $(100, 400)$, $(300, 600)$, and $(600, 300)$. Almost all runs resulted in orientation and positions within a few percent of the target, but $(100, 400)$ did not work very well for general navigation with rotations, because very small rotations were not generated reliably. In general, however, these results show promise for students to be able to use the techniques in this paper to achieve handy navigation to specified coordinates.

6 Conclusion

We have overviewed several ways in which students in the middle school to high school range can use algebra, data analysis, and trigonometry to compare motion planning strategies for a typical educational robot and program general routines to navigate by dead reckoning. Some qualitative observations are that routings considered in Section 4 are shorter and/or have fewer velocity discontinuities, but the horizontal and vertical routings considered in Section 3 are simpler to implement. Further, navigation is faster with swings than with rotations when routing horizontally and vertically, though rotations are generally a better basis when routing along more of a straight-line path. The results also show the expected result that navigation time generally increases with the track width.

A valuable next step would be to create worksheets directed towards students to help them work through the types of analyses in this paper. It would also will be nice to incorporate consideration of the case in which the orientation (angle) of the robot is to be changed in the final position relative to the start position.

References

1. Balkcom, D.J., Mason, M.T.: Time optimal trajectories for bounded velocity differential drive vehicles. International J. of Robotics Research **21**(3), 199–217 (2002)
2. Ben-Ari, M., Mondada, F.: Elements of Robotics. Springer-Verlag (2018)
3. Chitsaz, H., La Valle, S.M., Balkcom, D.J., Mason, M.T.: Minimum wheel-rotation paths for differential-drive mobile robots. The International Journal of Robotics Research **28**(1), 66–80 (2009). https://doi.org/10.1177/0278364908096750
4. Greenberg, R.I.: Pythagorean approximations for LEGO: Merging educational robot construction with programming and data analysis. In: Proceedings of the 8th International Conference on Robotics in Education, RiE 2017, *Advances in Intelligent Systems and Computing*, vol. 630, pp. 65–76. Springer-Verlag (2017)
5. McGee, S., Greenberg, R.I., Reed, D.F., Duck, J.: Evaluation of the IMPACTS computer science presentations. The Journal for Computing Teachers pp. 26–40 (2013). International Society for Technology in Education, http://www.iste.org/resources/product?id=2853
6. Thiruvathukal, G.K., Greenberg, R.I., Garcia, D.: Understanding turning radius and driving in convex polygon paths in introductory robotics. http://ecommons.luc.edu/cs_facpubs/202 (2018)
7. Wang, H., Chen, Y., Souères, P.: A geometric algorithm to compute time-optimal trajectories for a bidirectional steered robot. IEEE Transactions on Robotics **25**(2), 399–413 (2009)