
Evolution by Genetic Programming of a Spatial Robot Juggling Control Algorithm

Stewart N. Taylor
Electrical Engineering Department
McCullough 150
Stanford University
Stanford, CA 94305

Abstract

In the last few years, a handful of researchers in the fields of robot control and artificial intelligence have fixed upon robot juggling as an interesting problem. Some progress has been made toward a solution and a few successful implementations for this control task have been developed, but the success of the known algorithms is as yet not fully understood. This is a complex control problem with inobvious criteria for success. As such, it represents an interesting challenge for evolution-based algorithms, an application which tests their supposed power and flexibility. It is the goal of this project to (re-)discover, using the genetic programming method, as many of the principles involved in robot juggling as possible. In doing so, it will focus on one particular robot juggling task, that of keeping one ball aloft by hitting it with a hard paddle, and a particular, "real-life", robot model.

1 INTRODUCTION

This paper documents an attempt to evolve a control program for robot juggling. What is meant in this paper and elsewhere by robot "juggling" is not juggling in the human sense. Conventional human juggling involves two hands and three balls; a person begins by holding three balls, two held in one hand and the last in the other hand. One ball from the former is thrown aloft, and thereafter each hand releases the ball in its grasp right before catching the next ball. The stability of the system depends on the timing of the throws, in that no throw may follow the previous too closely or else catching the throws becomes too difficult.

In reality, existing robots are far more adaptable to the task of batting a ball than catching and throwing one; this task, when performed with one ball, is more closely akin to bouncing a tennis or ping-pong ball on a racket. Both the calculations and the timing (for the grasping motion) are simpler. Since for robots the difficulty of stable control, especially real-time algorithm development and execution, is a greater restriction than the physical or sensory constraints, the less intricate "batting" is preferred over the more intricate "catching".

The juggling task is then described as keeping one or more balls aloft for a period of time by hitting it with a surface of some sort. The stability of the system depends only on the force and direction of each impact. This task requires an algorithm that will intercept the ball as it falls, and hits it back upward with some force and aimed in such a way that the ball will fall within reach of the robot again. It's this algorithm that is the intended product of the genetic programming. By guiding the development of a population of LISP-like functions, one hopes to obtain a series of programs that are progressively better than the last at controlling this robot arm in the act of juggling.

2 BACKGROUND

(Rizzi 1992a), (Rizzi 1992b), (Rizzi 1993), and (Rizzi 1994) describe a series of experiments with robot juggling performed by A. Rizzi and D. Koditschek. Using Yale's Buhgler Arm, they were able to keep one ball (Rizzi 1992a,b), and later two (Rizzi 1993) aloft and stably controlled for an arbitrarily long time. Figure 1 is a conceptual diagram of the Yale robot that performed this feat.

Rizzi and Koditschek developed for their robot arm controls a highly involved function based on four simple constraints:

- tracking the arm under the ball to achieve impact with the ball

- mirroring the vertical motion of the ball (to achieve a reasonable vertical rebound)
- following the radial motion of the ball by raising or lowering the paddle accordingly, and thus adjusting the aim of the paddle
- turning the paddle itself to compensate for tangential motion

Each of these corresponds to part of the expressions in their control function. The entire control function, however, is far more complex than would seem necessary to fit these constraints, because it is a full solution to the dynamic equations of the robot.

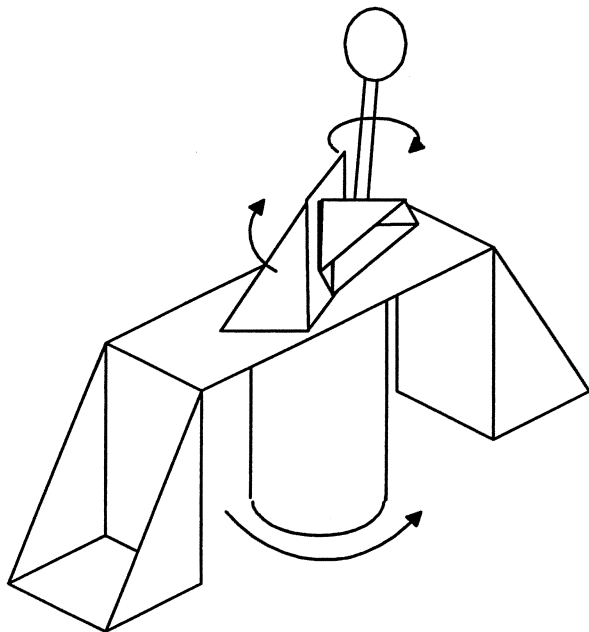


Figure 1: The Yale Buhgler Arm, after (Rizzi 1992a)

Schaal and Atkeson of MIT have also developed models for robot juggling. Though their research proceeds in a more theoretical direction, for example not taking into account the physical robot that performs the task, it is interesting here in that their solution should be applicable to other similar juggling tasks. In (Schaal 1993), they describe an "open loop stable control strategy"; that is, a method of control that does not require feedback to produce a stable result. In this case, they claimed that for a certain frequency and angle of attack, a paddle that merely oscillates in a sinusoidal manner:

$$x_p = A \sin(\omega t + \theta_0)$$

$$\theta_0 = \arccos\left[\frac{\pi g}{A \omega^2} \left(\frac{1-\alpha}{1+\alpha}\right)\right]$$

where θ is the phase, α is the coefficient of restitution, x_p is the paddle velocity, ω is the angular velocity, and g is gravity. The oscillating paddle can produce a stable bouncing in the ball by hitting it at regular intervals.

3 MOTIVATION

Like any robot control task, juggling is both complex and computationally intensive, mostly owing to the elaborate dynamics of the robot. Furthermore, it is a real-time task, adding a significant constraint to the model. As such, evolving the a solution using the full kinematic and dynamic model is perhaps too complicated to be achieved in a reasonable amount of time. However, even solutions evolved to a simpler model could serve a valid purpose.

The two goals of this project were to reproduce these results with a genetically evolved solution, and to note any previously unconsidered algorithms for robot juggling that might evolve. Accomplishing the former goal would validate that the method is a reasonable one; achieving the latter might provide some insight into the area of automated robot control (well, juggling anyway). It is not entirely unlikely that, since this is a new and relatively unresearched area, efficient algorithms exist that have as yet not been discovered.

4 METHODS

As previously mentioned, human juggling is considerably different from the usual robot juggling. However, initially this project was to explore the control required for a hypothetical two arm robot with human-like range of movement, and it was to perform the basic juggling task of three balls passing between the two hands. Unfortunately, there were a number of problems with this method as stated.

For one, even when reduced to two dimensions, the problem required six degrees of freedom: four joints and two hands, open or closed. Developing 6 independent outputs is not impossible, but the effect of training gets diffused over them. Even then, training is difficult without a very involved fitness function, just because success at juggling of this time doesn't reduce well to a single number.

These considerations partially drove the decision to use the relatively simple problem specification used by Rizzi and described above. That definition of robot juggling, a simplified version of that robot model were both adapted for use as a genetic programming problem.

4.1 ASPECTS OF THE PHYSICAL MODEL

Since the goal was to observe algorithms for robot juggling, rather than to generate an exact algorithm for actual use with a robot, any simplification of the model that did not affect the solution set too significantly was considered a benefit. Here are some notable simplifications:

- greatly simplified robot dynamics
- no limit on arm acceleration
- robot "hand" is circular (actual robot hand is rectangular)
- ball is considered to be a volume-less point
- within a time step (1ms), motion of hand and ball are linear

4.1.1 Time

Time is modeled discretely, in 1 ms increments; this is consistent with 1MHz sampling rate in (Rizzi 1992a). This places a lower bound on the period of a juggle, and reduces the accuracy of certain hand/ball impact calculations.

4.1.2 Ball Model

Balls were treated as points operating under gravity. They do not decelerate due to air resistance. The only random factor in the motion of the ball is the rounding error; other than that there are no modifiers introduced into the flight (e.g. a breeze) to make the simulation more realistic.

4.1.3 Robot Arm Model

Figure 2 is a diagram of the kinematics of the robot arm. The arm used in this experiment is the Buhgler Arm, as described in (Rizzi 1992b). The variables q_1 , q_2 , and q_3 are the orientations of the three joints of the arm. The distance d_1 is 1.0m, and d_2 is 0.0m.

The striking surface, the "hand", at the end of the arm is considered to be circular and 0.10m in diameter. The position of the center of the hand in 3-space is \mathbf{p} , and its surface normal is \mathbf{n} .

The position and unit normal of the hand can be calculated as

$$\vec{p} = \begin{bmatrix} \cos(q_1) \sin(q_2) \\ \sin(q_1) \sin(q_2) \\ \cos(q_2) \end{bmatrix}$$

and

$$\vec{n} = \begin{bmatrix} \cos(q_1) \cos(q_2) \cos(q_3) - \sin(q_1) \sin(q_3) \\ \cos(q_2) \cos(q_3) \sin(q_1) + \cos(q_1) \sin(q_3) \\ -\cos(q_3) \sin(q_2) \end{bmatrix}.$$

Note that the hand of a robot of this design cannot be moved to arbitrary coordinates. Its motion is restricted to points on a sphere surrounding the robot. This is a significant restriction for the robot, and a major

challenge for a program attempting to move it to meet a falling object.

The variables which a control program can modify are the angular velocities (dq_1 , dq_2 , dq_3) of the three joints. They are limited to maximum velocity of 8 RPS, but there is no maximum acceleration (other than $16 = 8 - (-8)$ RPS each time step).

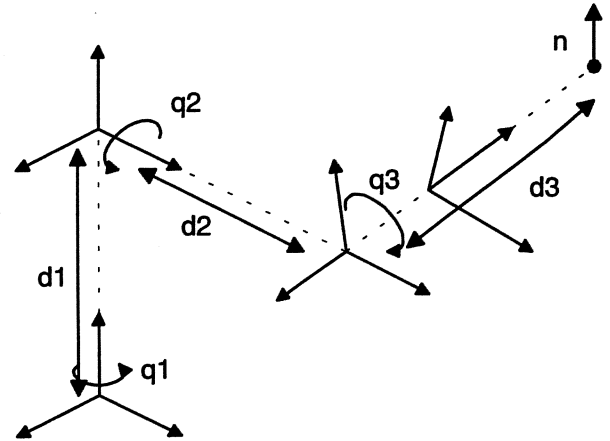


Figure 2: Kinematics of Buhgler Arm (Rizzi 1992a)

4.1.4 Impact Model

When the surface is detected to have hit the ball (or vice-versa), an impact takes place, and the ball rebounds in come way. The impact model is a standard model for inelastic collisions, described in (Rizzi 1992b) as

$$\dot{b}' = \dot{b} + (1 + \alpha)nn^T(v - \dot{b}),$$

where \mathbf{b} and \mathbf{b}' are the old and new ball velocity vector, \mathbf{v} is the surface velocity, \mathbf{n} is the unit normal vector of the surface, and α is the coefficient of restitution (set to 0.5). In essence, the components of velocity normal to the surface are inverted and multiplied by α , while the components parallel to the surface remain unchanged. In this model, the paddle is assumed to be considerably more massive than the ball, and thus the paddle velocity is unaffected by the collision.

4.2 GP PARAMETERS

Figure 3 is a tableau of the most noteworthy parameters to the GP employed. Following are explanations of the meanings of each entry, and the motivation for them.

4.2.1 Functions and Terminals

The functions that are used to construct a set of operations upon the terminal set to produce the results. It includes the basic arithmetic operations, Even though it results in a performance degradation, the function set

includes asin, acos, atan, and sqrt by necessity. Stability of the juggling system can depend on the precision of the calculation of the position of the arm relative to the parameters q1, q2, and q3.

Certain functions, such as division, and arcsin, are only defined for a restricted range of inputs. For the purposes of this experiment, special versions of these functions are used, with output values specified for the undefined regions. In the case of division, arcsin, and arccos, either 0.000000001 or -0.000000001 is substituted for the illegal input. In the case of sqrt, invalid (negative) input values are inverted.

Ball position is an obviously essential piece of information to the GP, and yet the cartesian coordinates (x,y,z) are a very raw form of data. Determining the robot arm angles, much less the delta thereof, to place the robot hand at a given (x,y,z) is in itself a complex task. Thus, one possible set of inverse functions is provided, inverse q1 and inverse q2 which give the angle settings that would produce a given (x,y,z) location in the hand. One formulation of the inverse p function, for q1, q2, and q3, is:

$$\vec{p}^{-1} = \begin{bmatrix} \arctan\left(\frac{-BallY}{-BallX}\right) \\ -\frac{\pi}{2} + \arctan\left(\frac{1.0 - BallZ}{\sqrt{BallX^2 + BallY^2}}\right) \\ q3 \end{bmatrix}$$

Note that in this formulation, q3 remains a free variable. Since, as previously noted, the robot hand may only be moved to points on a sphere, it really only has two degrees of freedom for position, hence the free variable. The third degree of freedom modifies only the orientation of the hand; this orientation is not completely restricted in range as well.

4.2.2 Results Produced

There are three outputs, dq1, dq2 and dq3, the angular velocities per time step of the joint angles. These outputs were limited to 8 revolutions per second (approx. 0.05 radians/time step).

4.2.3 Fitness

The fitness was calculated from the minimum distance from the hand to the ball and the time steps executed. The former acts early in the run, when most programs are unable even to find the ball. The later dominates when most programs have managed to hit the ball at least once; they are then trying to prevent it from terminating because the ball stops, hits the ground, or hits the ceiling (at 3m). Due to the constraints of time and the chosen fitness function, only one test ball was dropped for each fitness evaluation.

Objective:	To evolve with GP a program to control the arm of a "juggling" robot.
Function Set:	+, -, *, %, asin, acos, atan, sqrt, sin, cos (invq1, invq2)
Terminal Set:	Ball position:BallX, BallY, BallZ; Hand position:HandX, HandY, HandZ; Joint angles: q1, q2, q3
Results Produced:	dq1, dq2, dq3
Fitness Cases:	Up to 20000 time steps (20 sec)
Fitness:	minimum distance of hand from ball during run + (20000 - time steps executed)
Hits:	time steps executed before ball hits ground or ceiling or is "caught" by paddle
Wrapper:	dq1, dq2, dq3 limited to 8 revolutions per sec
Population Size:	1000 (500)
Termination Criteria:	20000 time steps executed / fitness = 0

Figure 3: Tableau for Robot Juggler

4.2.4 Population

For a problem of this complexity, even though the time to do each evaluation is significant, a population of 500 was determined to be a little small; for most runs, 1000 was used. Compared to the number of terminals and functions, and the very small number of combinations that would be at all viable (i.e., even hit the ball once), a population of a thousand is not that significant. Without that many individuals, the GP converged rapidly on bad solutions, such as even a bad "hold then hit."

4.3 IMPLEMENTATION

The implementation of GP used was Dave's Genetic Programming Code (DGPC), recompiled under 32-bit protected mode DOS with Symantec C/C++. All runs were performed on a 90MHz Pentium Processor-based computer. Each run required approximately 2-4 hours to complete.

5 RESULTS

As is often the case, the evolution of the GP usually took place in obvious steps, as the GP "discovered" a new way to handle the ball. Over a number of runs these categories of results were clearly repeated. The points of temporary equilibrium were:

- “miss ball entirely”

This was the result of most first-generation algorithms. The number of hits is equal to the time steps it takes for the ball to fall straight to the ground.

- “hit ball at random”

Frequently an algorithm will be able to reach a ball, then hit it in some random direction that is then unreachable for a second hit. If the ball is hit at all upward it takes it longer to reach the ground.

- “hold steady beneath ball”

A method that was fairly effective was to wait directly beneath the ball, with palm facing up. The ball would bounce three times and then be considered by the program to have stopped and the run would end. This simple algorithm was nevertheless as effective as a poorly implemented “limited mirror.”

- “hold then hit”

One might hope that the next step after the “Hold Steady Under Ball” would be some sort of small movement in the z direction for the paddle, perhaps in time with the rise and fall of the ball. Unfortunately, the program result that was most favored by the fitness function was the “Hold then Hit” algorithm. After all but catching the ball on the paddle, the robot produces a velocity spike which throws the ball up in the air, and off out of reach in the x direction as well. Figure 4 is a sample of the code which produced this solution. Figure 5 is a graph of the ball and paddle position over time for this solution.

```
RPB_Num 0: (* HANDX (+ Q1 Q3))
RPB_Num 1: (acos Q2 )
RPB_Num 2: (sqrt BALLX )
ADF_Num 0: (- ARG0 (- ARG1
(- ARG0 (- ARG1 (asin ARG0 )))))
ADF_Num 1: (* ARG2 ARG2 )
ADF_Num 2: (* (acos ARG0 )
(+ ARG2 (% ARG1 ARG0 )))
```

Figure 4: Code for “Hold Then Hit” Solution

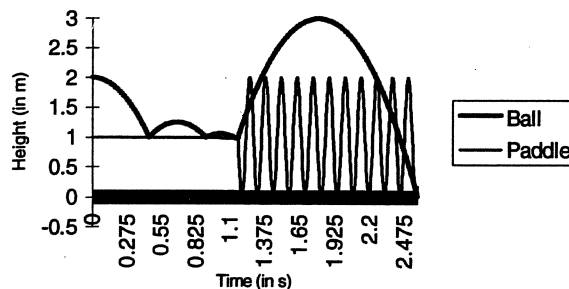


Figure 5: Graph of Position vs. Time for the "Hold Then Hit" Solution

- “limited mirror”

When the inverse q1 and inverse q2 functions were added, which all but gave the answer away, some solutions that was produced were limited mirrors. The mirrors would either produce too much rebound or not enough, and eventually the juggled ball would hit the ceiling or stop bouncing on the paddle.

- oscillation

Using sin() the GP was capable of evolving a limited oscillator. The oscillations were quite small, and for the most part took place in the xy plane. However, there was enough angle on the paddle to create a small upward force that was enough to juggle the ball 1-2 mm above the paddle. Figure 6 is a sample of the code which solved the problem in this way.

```
ADF_Num 1: (* ARG1 ARG0 )
RPB_Num 0: (ADF0 (% HANDZ HANDX
)(asin HANDX ))
RPB_Num 1: (* (* BALLX Q2)(cos (* Q2
HANDZ )))
RPB_Num 2: (cos (* Q2 HANDZ ))
ADF_Num 0: (* ARG0 dT )
```

Figure 6: Code for Oscillating Solution

6 DISCUSSION

The GP could be said to have had limited success with the juggling problem. Most runs ended with a “hold then hit” solution, a solution that seems unusual and unlikely. Moreover, that solution would probably not succeed if there were better controls on the radial acceleration of the arm parts. It is possible that these were caused by some loophole or singularity in the floating point functions, or some unconscious bias in the selection of functions and terminals.

It did "discover" the two previously described techniques for juggling, though each in a somewhat limited manner. It was able to find the oscillation technique, but the oscillations it used were so small as to be undetectable on a typical line graph. It did produce some mirror-like behaviour, but this required special functions within the function set of the GP. Some behaviours on the list, radial and tangential velocity compensation, for example, would not be expected with the limited fitness cases that were used; nevertheless, this should not have prevented the GP from producing a stable (20 secs.), reasonable (above 1mm) juggling program.

One possible culprit was the fitness function. Although a very high fitness indicated success at the task, the levels intermediate fitness values did not necessarily reflect proximity to the solution; that is, there was significant deception inherent in the fitness function. The "hold then hit" is an example. It is certainly less of a solution than an over-eager mirror program that hits the ball too hard and into the ceiling. The mirror program might only need a little tweak to be effective, but would probably not get much chance to reproduce itself.

A perhaps more significant consideration would be the population size. Considering the intricacy of the required task, there is probably not enough genetic information in a mere 1000 individuals to be reasonably expected to solve this problem. It seems likely that a starting population on the order of ten times that number of individuals could produce the exact solution.

7 FUTURE WORK

The number of fitness cases was insufficient to prevent the GP from specializing beyond usability. While usability was not the focus of this project, nevertheless the more general the solution the more applicable it might be to this problem in other environments.

As noted in the discussion, the fitness measure is subject to a fair amount of deception; it was likely the main weakness in this GP formulation. The "hold then hit" program is a good example. It contains little valuable genetic information, since its internal operation is not much like the known juggling algorithms. However, it has a fairly high fitness with the existing fitness measure, because it is as fit as the "hold steady" programs, plus it holds the ball above the ground significantly longer by hitting it hard at the last minute before the ball stops. A better fitness might reward the program according to how close each bounce of the ball is from a good juggling throw/hit. This would help avert the local minimum problems, but could also be "meddling" to put too much human bias into a run.

A more realistic robot model was outside of the scope of this particular project. The acceleration of the joints was assumed to be instantaneous, and was limited only by the angular velocity limit. More accurate dynamics such as those in (Rizzi 1992b), would be very complicated to

implement, and also probably quite slow; however, it would limit the degree to which the GP can exploit the "loopholes" in the physical model.

Once the other issues are resolved, the other goals will be to tackle more general single-ball juggling cases and then the two-ball problem as in (Rizzi 1993). Unfortunately, the fitness measure which was just adequate for this problem, flight time, is likely to be even less efficient at producing good strategies for two-ball juggling.

8 CONCLUSIONS

The GP was able to reproduce at least one of the algorithms for robot juggling described in the literature, the open-loop "vibration" control strategy. A mirror rule was also achieved, but with the aid of two tailor-made functions.

This was a particularly difficult robot control problem. The biggest obstacle was attempting to "encourage" juggling without really biasing the evolution toward one solution or another. Until the number of time steps reached 3000-4000, many other behaviours that bore no resemblance to juggling achieved equal fitness. The fitness function proved very important, and the one chosen was not sufficient.

Still, the GP was able to reproduce existing algorithms somewhat, even with a poor fitness function; it is possible that new algorithms could be developed, given sufficient time and results analysis. The applications of the GP to robot juggling warrant further work, and would be facilitated by with improvements to the physical model, and the GP parameters, especially fitness.

Acknowledgements

John Koza provided considerable encouragement and advice, and a great GP textbook, and as such was instrumental in this project and subsequent paper.

References

- Koza, J. Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA: The MIT Press 1992.
- Rizzi, A. A. and Koditchek, D. E. Further progress in robot juggling: solvable mirror laws. In Proceedings of the 1994 IEEE International Conference on Robotics and Automation. Piscataway, NJ: IEEE 1994. Pages 2935-2940.
- Rizzi, A. A. and Koditchek, D. E. Further progress in robot juggling: the spatial two-juggle. In Proceedings of the IEEE International Conference on Robotics and Automation. Piscataway, NJ: IEEE 1993. Pages 919-924.
- Rizzi, A. A. and Koditchek, D. E. Progress in spatial robot juggling. In Proceedings of the 1992 IEEE International Conference on Robotics and

Automation. Piscataway, NJ: IEEE 1992. Pages 775-780.

Rizzi, A. A., and Koditchek, D. E. Distributed real-time control of a spatial robot juggler. *Computer*. 25(5) 12-24. 1992.

Schaal, Stefan, and Atkeson, Christopher G. Open loop stable control strategies for robot juggling. In *Proceedings of the IEEE International Conference on Robotics and Automation*. Piscataway, NJ: IEEE Service Center 1993. Pages 913-918.

National Resource Laboratory
for the study of
Brain and Behavior

**Proceedings of the Workshop on
Genetic Programming: From Theory
to Real-World Applications**

Justinian P. Rosca, Editor

Technical Report 95.2
June 1995

UNIVERSITY OF

ROCHESTER
