# Control System

## Speed

In choosing a processing unit for the control system, the criteria most concerning the device are speed, programming difficulty and cost. The table below highlights the options available in a general sense.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Programming  Difficulty | Speed | Power | Cost | Display |
| 8/ 16 bit microcontroller | Moderate | Fast  ~20MHz | Low  <1W | ~$50 | Difficult to add |
| 32 bit microcontroller | Moderate | Very fast  ~600MHz | Low  1-5W | ~$150-400 | Possible |
| 32 bit x86 processor (PC) | Easy/moderate | Slow/fast  (depending on OS & h/w) | High  <50W | ~$150-500 | Easy to add |
| FPGA | Moderate  -difficult | Very fast  ~100MHz (no overhead) | Low  <5W | ~$100-$800 | Difficult to add |

Table - Processing Unit Options

How fast of a processing unit do we need? We know that the system as a whole must update at a rate of at minimum 1 kHz based on the results from Nikolai’s thesis. There are two main tasks the processing unit must perform. It must first read the values from the encoders/sensors and calculate the position of the end-effecter relative to the virtual and physical surface. Then based on that position, it must calculate the position of the hard restraint mechanism. In other words, it does a forward kinematics calculation and then a reverse kinematics calculation.



Figure - Block diagram depicting processing blocks in green

Determining an exact metric for speed is difficult to do because speed depends on not only the frequency of the processing unit but also the architecture. However, if we assumed the architecture allowed that all instructions completed in one cycle, then the minimum speed can be determined from the code excerpt below.

x\_position = Length\_3 - (Length\_4 \* sin(Theta\_4));

x\_position\_neg = Length\_3 + (Length\_4 \* sin(Theta\_4));

delta\_z = Length\_4 \* (1 - cos(Theta\_4));

Length\_2\_star = (Length\_2 ^2 + delta\_z ^2) ^.5;

Theta\_star = tan(delta\_z / Length\_2) ;

Theta\_12\_star = Theta\_12 + Theta\_star ;

y\_position = Length\_2\_star \* cos(Theta\_1 + Theta\_12\_star) - Length\_1 \* sin(Theta\_1) ;

z\_position = Length\_2\_star \* sin(Theta\_1 + Theta\_12\_star) + Length\_1 \* cos(Theta\_1) ;

Figure - Code excerpt from Matlab that calculates tool position

With the assumption that floating point instructions complete in the same time as scalar instructions, all scalar instructions must be accounted for. It can be safely said to increase the number of instructions by an arbitrary factor of 200 to account for overhead and other instructions.

So a processor running at 5.6 MHz would be fast enough but only a general purpose processor like a Pentium processing core has the architecture – specifically a FPU (float point unit) – that would satisfy our assumption. However, Pentiums and alike have frequencies of 1GHz and above; so therefore any processor in that class would work for our purposes.

What about the worst case scenario? For this analysis, the focus is on the microcontroller used by last year’s group – the Atmel AVR 8-bit AT90CAN128 running at 16MHz. The code excerpt is used again as a comparison. Since this microcontroller does not have a floating point unit, all floating point instructions must be converted into integer math by the compiler. This has significant impact on performance, which can be seen in the benchmark below depicting a simulation for a “sine” instruction. If the code excerpt were to be implemented on this microcontroller, it would not satisfy the 1kHz speed requirement. However, simple floating point math like add and subtract have less of an impact on performance. In the code excerpt, 7 of the 28 functions are trigonometric functions. If we assume an average time of 25us and similar amount of instructions for the reverse kinematics calculation, the microcontroller is still not fast enough.

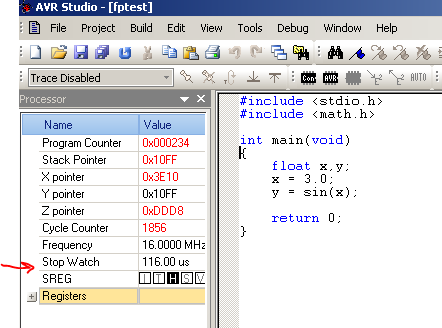


Figure - Floating point benchmark on AT90CAN128



Figure - Frequency estimate

There are several ways which the AT90CAN128 can be used. One method is to pre-compute all the possible values of the end-effecter position with corresponding block position and use a lookup table to search for the blocker position. This method is at the expense of resolution limited by the memory available.

From these results, it can be concluded that a faster 32-bit processor should be pursued if the cost is not an issue. Other factors like programming difficulty and real-time operating systems require further analysis.

## Accuracy

The accuracy of the robot is determined by the amount of error produced by the control system from the input to the output. The following diagram depicts all the possible errors involved.

encoder

Compute

position

Error 1

Compute

blocker

Error 4

Error 3

Error 2

Motor controller

motor

The sum of all the errors must be less than the equivalent of 0.5mm as per the evaluation criteria. Error 1 and 4 are fixed based on the encoder and motor controller chosen for the design. For the purpose of this analysis, the device as a whole has negligible play or slack. The main sources of error are from the mechanical devices. An error due to the resolution of the encoder will be amplified when a position value is computed, which is further increased on the way to the motor due to the resolution of the motor controller. The error of computation is negligible if floating math using no lookup tables is assumed. IEEE 754 32-bit floating point specifications provide accuracy of 7 digits with an exponent of -95 to 96.