Intelligent Machines Design Laboratory

C.A.T
Can-crushing
Autonomous Terror

University of Florida
Dr. Keith Dotty
Jeanine Shevlin
28th April 1997

TABLE OF CONTENTS

ABSTRACT

C.A.T., standing for Can-crushing Autonomous Terror, was designed to seek out a person who needs to get rid of a coke can, crush the can and deposit it elsewhere. There are four behaviors which C.A.T. exhibits while accomplishing its goal: obstacle avoidance, IR following, can crushing and depositing. C.A.T.'s design makes it rugged and able to withstand the force of crushing a coke can. The code used to program the autonomous behaviors was Interactive C.

EXECUTIVE SUMMARY

C.A.T. is an autonomous robot that was design to crush and deposit coke cans. In order for this to be done the major hurdle that had to be overcome was finding the mechanical advantage in order to produce the force necessary to crush the can. By doing static tests, it was discovered that 50 lbs will flatten a dimpled soda can. Otherwise, without dimpling the can can withstand around 500 lbs of force. Many options were explored in order to accomplish this goal and the method finally settled upon was using a power screw. With a power screw you can obtain high ratios of mechanical advantage, plus they are easier to construct than other Kinematic mechanisms. The power screw used was an .5" diameter and 10 threads/in ACME threaded screw. The motor used to power it runs at 40 rpm at 15 Volts When stalling the motor it draws around 2.5 amps. Because of this draw on amps, a relay was connected between the motor and the LED port. Also the motor requires it's own battery pack.

C.A.T. was designed to be able to follow an IR beam and to avoid obstacles. Since both of these behaviors use the same set of sensors the programming was a little tricky. In order to accomplish the arbitration, I had a process that continuously turned on the LED's and took sensor readings, then turned them off and took sensor readings. These values after being compared were used to determine if: 1. There was an obstacle, 2. There was an IR beam, 3. There was both, or 4. There was neither. In the arbitrator a series of if-then statements were used in order to control the motors.

The platform designed to be C.A.T. although sturdy and made out of wood, is heavy for a robot. The motors and the wheels are under enormous stress when the robot is both wheeling around and crushing a can. The spacers designed to attach the motor shaft to the wheels helps to keep the shaft from snapping right off of the wheel. It also helps the wheel from bending under the load.

INTRODUCTION

The idea for C.A.T. came while I was sitting in another class drinking a coke. I noticed that there were cans lying around the room that were left for the janitor to collect. I thought that it would be helpful if there was a robot designed to go around, crush cans and deposit them in a designated area. Of course this design will not work in areas with stairs and other such uneven levels. But for a flat tiled or carpet room this robot will basically tidy up a bit.

In order to accomplish its goals C.A.T. was designed to follow an IR beam to a remote, so a person can insert the soda can. This was not the original plan, but this way a person could be sitting on their couch, drink their soda and then decide to dispose of it. Another beacon is set up somewhere for the robot to drive to once it has crushed its can, to deposit it.

This paper discusses what has been accomplished on C.A.T., starting with the integrated system and ending with the C.A.T.'s behaviors. Each component of C.A.T. is described in great detail with diagrams and figures to fully explain the robot's purpose.

INTEGRATED SYSTEM

To achieve C.A.T.'s overall goal, it's design uses a variety of sensors and system upgrades. The 68HC11 board was the backbone of C.A.T.'s operations. Figure 1 shows a flow chart of the processes.

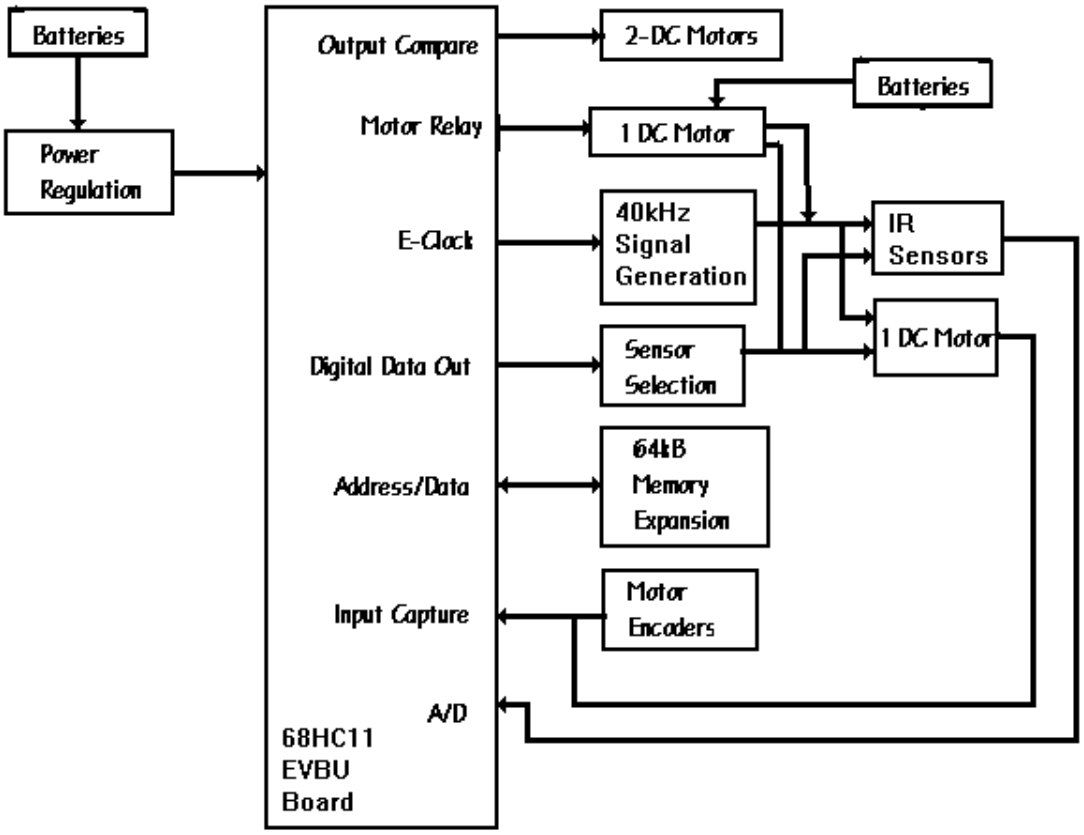

Figure 1

The specifications for the robot are that it includes four different behaviors and a variety of sensors. The four behaviors exhibited by C.A.T. are: light following, obstacle avoidance, can crushing and can depositing. The sensors used are IR sensors and bump/limit sensors.

MOBILE PLATFORM

The mobile platform for the robot is made out of ½" plywood. It is a 11" diameter circle with a rectangle cut out of the front. The rectangle is 3.5" wide and 4.5" deep. See the plans in Appendix A. This cut out is where the can will be crushed and deposited. The 11" diameter is needed because there is a 4" aluminum square sliding base under the platform and also two motors that are 2" wide each. There are a pair of tracks attached under the platform for the base to slide back and forth on. The wheels are placed directly on the center axis of the platform and are attached by the motors. The ME11 and MC68HC11 boards are attached to the back of the platform with 2 ½" #6-32 screws. There is a bridge built over the boards attached to the platform at the side of the boards. The bridge serves two purposes: decoration and as a platform for the switches to be attached to.

Caster wheels were added to the platform in two areas. The first area was behind the motors in order to prevent C.A.T. from flipping backwards. The second areas was underneath the front of the sliding base tracks. This prevents the tracks from coming apart while the can is being crushed. The caster wheels are attached to the tracks with blocks of wood so as to tilt the platform back slightly. This offsets the center of mass, preventing the robot from toppling over during the crushing process.

During the design of C.A.T. I learned many lessons about the structure of the platform. If I had it to do over again, I would not have put the robot on wheels. Instead I would have made it on tracks, like a tank. This would provide more sturdiness while the can is being crushed. I noticed that the wheels were starting to bow out during the crushing process. Also I would have made the entire robot out of aluminum. This would be light and strong. With the wood I kept having problems with it splitting as I was drilling holes.

ACTUATION

Wheels

The purpose of the two wheels are to drive the robot. Each wheel is controlled by a separate DC motor. These motors are driven directly by the 6811 board. The two motors require between 3 and 4.5 volts each and have a gear ratio of 64.8:1. In order to receive only the necessary voltage, a wire connected from the batter pack's 4.6 volt cell to the motor driver chip was soddered in place. The speed and torque of the motors vary depending on the voltage you apply to them. The motor shafts are attached to the wheels by a spacer I designed, with the help of Steve Sowa (Mechanical Engineering Lower Machine Shop Supervisor), and had machined. The plans for this spacer are in Appendix B. The motors are controlled by the code and are dependent on whether C.A.T. is in follow mode or obstacle avoid mode. When following, the motors turn the wheels to move toward light. For example if the light is coming from the robot's right, then the right wheel is stopped and the left wheel keeps turning. When in avoid, the motors turn the wheels to avoid the obstacle. This is done in a manner opposite to that described above. The code for the motor controls are included in Appendix C.

Sliding Base

The sliding base moves by means of a power screw. The motor turns the screw which in turn moves the plate in the direction of the threads. This happens because a nut is affixed to the plate. The motor turns in one direction until the base hits a switch at one end of the track. The base waits in the extended position until the can is crushed and C.A.T. has driven to the designated drop off area.

The power screw motor operates at approximately 45 rev/min and has 42 oz. in of torque. In order to run this motor another motor driver chip was added to the EVBU board. The schematic for this setup is shown in Figure 2.

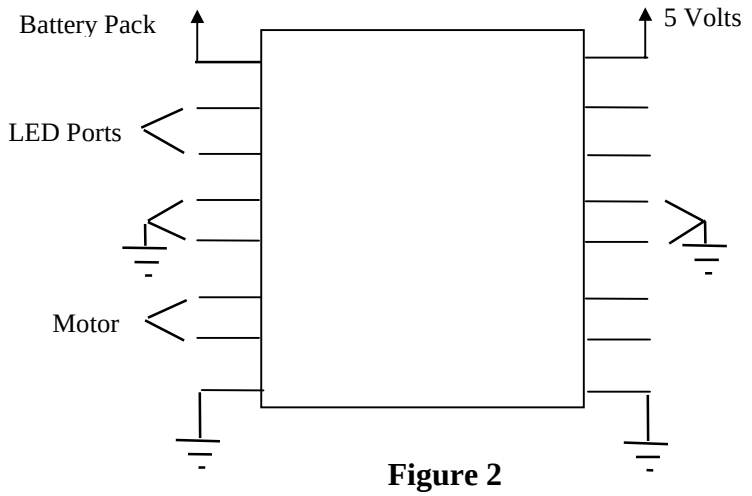

Figure 2

The brass screw was chosen because it is a softer metal than steel and thus does not require much lubrication. The ¼ x 20 size was chosen for its weight, one size larger would move the plate quicker but weighed about twice as much. Having the 20 threads/in instead of 16 does not affect the performance of C.A.T.. The 20 threads/in screw along with the 45 rev/min motor allows the base to move 2.25 in/min. While the 16 threads/in moves 2.81 in/min. These figures were reached using the following calculation:

$$45 \text{ rev/min} \times (1 \text{ in} / 20 \text{ rev}) = 2.25 \text{ in/min}$$

Since the plate has to travel about 2 in to dispose of the can it will take a little less than a minute to complete the task. This can be sped up if a faster motor was used. A schematic of the sliding base is attached as Appendix D.

An interesting lesson that I learned with the sliding base is that JB Weld epoxy doesn't hold up as well as I had hoped. The sliding base is one of the first additions I made to C.A.T. For about two months I was sliding the base back and forth. Unfortunately the night before C.A.T. was to have its final demonstration the epoxy broke loose. Next time I will use a steel screw and a steel plate so the they can be welded together.

Can Crusher

The can crusher moves in the same manner as the above mentioned sliding base. Only in this case the screw is pushing a plate downward with a greater force. Before any device could be built, the motor and screw had to be sized.

Appendix E includes the calculations performed to determine how much torque the motor would have to output in order to crush the coke can. The calculations are based on the use of an ACME threaded screw with a .5 in diameter and 10 threads per in. The reason for this choice is its efficiency. A regular machined screw has about 33% efficiency, while an ACME screw has 66%. An ACME screw has square threads and a different angle cut than that of a machined screw. For the best efficiency one should use a ball screw, however these can be very expensive. The results of the calculations show that the motor would need to output 14lb*in of torque in order to crush the can. The motor I purchased runs at 40 rpm and has high torque. Its original purpose is for a car window. Although 40 rpm is slow, you must sacrifice speed for torque.

In order to run this motor off of the EVBU it had to be hooked into the LED ports. However, since the motor needs to run off of 15V and draws between 1 and 2.5 Amps, a relay circuit had to be hooked up to the board. The schematic for this is attached in Appendix F. The parts were purchased at Radio Shack.

The cage for the can crusher is made out of wood and is screwed into the base. The motor is attached on top of the cage and screwed in. A coupling was designed and made out of a steel cylinder to attach the motor to the screw. This coupling was welded to the screw and attached to the motor with epoxy and set screws.

The hardest part of designing the can crusher was holding the motor down. Since for every force there is an opposing force, the motor kept wanting to lift off of the cage as it encountered high forces from the can. At first I screwed it down and had one strap going across the long side of the motor. This however was not strong enough to hold down the motor. Therefore, I took a bendable metal sheet and strapped it across the other end of the motor and screwed it into the sides of the cage. This allows the motor to have enough play so as to

not bind up the screw and yet hold it down to allow the total force to act upon the can. Figure 3 displays the strapping setup for the motor.

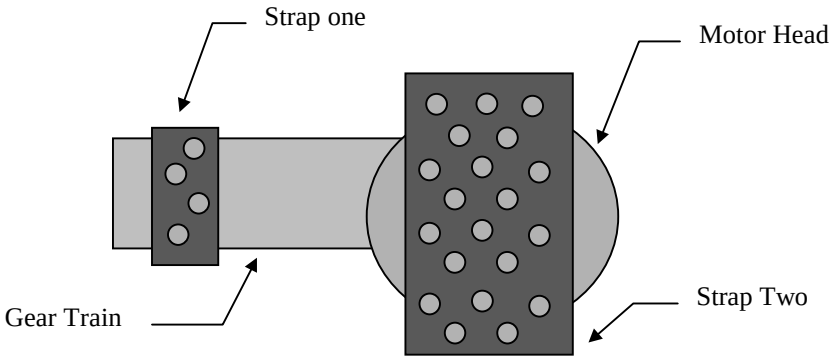

Figure 3

SENSORS

IR Sensors

For the behaviors of obstacle avoidance and IR following, I used IR sensors. While C.A.T. is roaming around it will either avoid obstacles or follow IR coming from a remote source. For the IR detector I used two hacked 40 KHz sensors purchased from NovaSoft. The IR LED's were collimated and affixed on the platform above the sensors. The sensors were attached on the front of the robot at a slight angle outward. The code in Appendix C demonstrates how C.A.T. arbitrates between the two behaviors. In the code I would turn the LED's on and take an analog reading, then turn them off and take a reading. Next I would compare these values to determine if there was an obstacle in front of C.A.T. or a beam from the remote. The following table displays the analog results from testing.

Analog Port	No LED/No Remote	LED on/Remote off	LED off/Remote on
2 (left sensor)	85	90-125	109-135
3 (right sensor)	85	92-130	105-130

Table 1

A lesson I learned from this sensor is that in the future to have more than two detectors. The main reason being that objects get past the detectors and the robot can easily be trapped. Another alternative is to program randomness into the code, so the robot turns a different way each time it encounters an obstacle.

Limit Switches

For both the sliding base and the can crusher C.A.T. is equipped with two kinds of limit switches to limit the movement of the plates. In the case of the sliding base two bump switches were used, purchased from NovaSoft. They were glued on the track, one in front and one in back. The circuit used to hook up the two switches to an analog port on the EVBU is pictured in Figure 4. The resistors used (24K, 11K, 10K) are chosen because when the switch that is connected directly to the 24K resistor is hit the analog port will receive a number almost twice as high as when the switch connected to the 10K resistor is hit. Thus the program can wait for the selected analog port to read a number in a certain range to act.

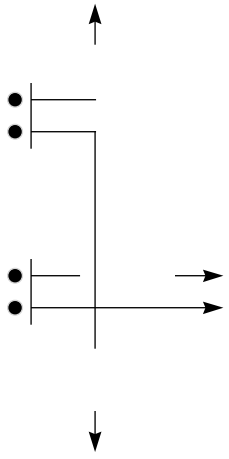

Figure 4

A hard lesson learned with this circuit was that sometimes analog ports do not receive steady readings. This can be due to noise in the circuit. In order to prevent this a diode was hooked between the analog port and ground.

The limit switches used for the crusher were roller switches purchased from Radio Shack. They were used because they are bigger, stronger and more durable. The circuit used for these switches is the same as in Figure 4. The only difference is that the resistors used were 2.2K, 1K and 1K.

BEHAVIORS

C.A.T. demonstrated four different behaviors: obstacle avoidance, IR following, can crushing and can depositing. Most of these behaviors have been touched upon in the above portion of the report.

Obstacle Avoidance & IR Following

The objective of these behaviors is for C.A.T. to reach its destination points without running into obstacles. By allowing C.A.T. to follow a remote it can be controlled by the user who wants to get rid of a can. But by programming in obstacle avoidance, C.A.T. can also run by itself, without the guidance of a remote. The code for this arbitration is as mentioned before attached in Appendix C, the two functions are follow() and obstacle_avoid().

Can Crushing

The objective of this behavior is also the overall objective of the robot, to crush a coke can. The program has the plate crush down to a certain height (until it hits the limit switch) and then stop. Before it crushes again, the plate is raised until it reaches the other limit switch. The code for this process is in Appendix C and is the function Can_Crush().

Can Depositing

The objective of this behavior is to get rid of the can once it has been crushed. Once C.A.T. has crushed the can it looks for a beacon to head to. It will wheel around for a certain period of time or until it reaches the beacon and then slide the base back to deposit the can. The function used to deposit the can is the

same as for sliding the base forward to allow for crushing, slide_base(). This function can also be viewed in Appendix C.

For all of these behaviors the hard lesson learned was programming. The programming for C.A.T. is different from any other kind of programming since it is not linear. This posed many problems for the programmer trying to debug the problems C.A.T. was having.

CONCLUSION

C.A.T. was functioning by demo day and performed each of its behaviors fully. The only problem it had was doing all four behaviors in one program. For some reason, the program would skip over the can crushing routine and move on to the next step in the process. In order to demonstrate that C.A.T. could crush cans I wrote a separate program for it to run. For someone who lacks programming experience I was extremely excited with the results produced.

Areas that need some work, would be the speed in which the base slides and the can crushes. Although one could say that the slow speed adds drama to the can crushing, one could also yawn while the process is taking place. Therefore, a faster method of crushing the can would be nice. This would require a faster, yet still has powerful motor, meaning more money needs to be spent.

If I started the project over I would have put the platform on tank tracks instead of wheels. As mentioned before this would provide a firmer base to carry the weight of the robot. If I had more time I would also like to put a can dimpler on C.A.T. so not only would it crush the can, but it would squeeze in the sides first as well.

For future work, I would like to set up a series of beacons so that after C.A.T. crushes a can, it can work its way through a house environment to the recycle bin. This way it will drive to one beacon, then to the next and so on until it reaches its destination point. I also would like to perhaps ad a gripper onto the platform so C.A.T. could also pick up the cans.

In conclusion I can not stress enough the wealth of knowledge that I have walked away from this project with. C.A.T. was a success and every step it took to get there provided insight into the design and building of autonomous robots

APPENDIX A

APPENDIX B

APPENDIX C

```
/* analog variables */
int onir_left, onir_right, offir_left, offir_right;
int left_dif, right_dif;
int threshold=90;
int thresh2=100;
int turn_off=0;
float half=100.0;
float n_half=-100.0;
int follow1=0;
int following=0;
int got_can=0;
/*variables for motor control*/
float fol_left, fol_right, obst_left, obst_right;
/* variables for arbitration*/
int deposit_can; /*initially set false*/
int crush; /*initially set false*/
/*********************************************************/
/* calculates the absolute value of an equation       */
int abs(int equation)
{
  if (equation >= 0)
     equation = equation;
  else if (equation < 0)
     equation = 0 - equation;
```

```
}
/*************************************************************/
void wait(int milli_sec)
{
  long timer_a;
  timer_a=mseconds() + (long) milli_sec;
  while(timer_a > mseconds()) {
    defer();
  }
}
/*************************************************************/
void read_ir()
{
 while(1)
 {
   while (turn_off!=0)
      defer();
   poke(0x7000,0x1);  /*turn IR on*/
   wait(15);
   onir_left=analog(3);
   onir_right=analog(2);
   poke(0x7000,0xC0); /*turn IR off*/
   wait(15);
   offir_left=analog(3);
   offir_right=analog(2);
```

```
    left_dif=offir_left-onir_left;
    right_dif=offir_right-onir_right;
  }
}
/*********************************************************/
void avoid_obstacles()
{
  while(1)
  {
    if (onir_left>threshold && onir_left>=onir_right)
    {
      obst_left=half;
      obst_right=n_half;
    }
    else if (onir_right>threshold && onir_right>=onir_left)
    {
      obst_left=n_half;
      obst_right=half;
    }
    else
    {
      obst_left=speed;
      obst_right=speed;
    }
  }
}
```

```
/*********************************************************/
void follow()
{
  while(1)
  {
    if ((offir_left>thresh2) && (offir_left>=offir_right))
    {
      fol_left=0.0;
      fol_right=half;
    }
    else if ((offir_right>thresh2) && (offir_right>=offir_left))
    {
      fol_left=half;
      fol_right=0.0;
    }
    else
    {
      fol_left=speed;
      fol_right=speed;
    }
  }
}
/*********************************************************/
/* this process slides the base back and forth to deposit */
/* the sode can                                           */
```

```
void slide_base()
{
  turn_off=1;
  poke(0x7000,0x0);

  if (deposit_can == 1)
  {
    deposit_can=0;
    poke(0x7000,0x18); /*send base backward*/
    while (analog(1) < 130)
      ;
    poke(0x7000,0x0); /*stop base*/
    wait(15);
  }
  else
  {
    deposit_can=1;
    poke(0x7000,0x10); /* send base forward*/
    while (analog(1)<75)
      ;
    poke(0x7000,0x0); /*stop base*/
    wait(15);
  }
  turn_off=0;
}
/*********************************************************/
```

```
/*this process moves the plate to crush the can          */
void crush_can()
{
  turn_off=1;
  poke(0x7000,0x0);
  if (crush==0)
  {
       crush=1;
       poke(0x7000,0x04); /*set plate crushing*/
       while (analog(4) < 120)
         ;
       poke(0x7000,0x0);
       wait(15);
  } /*end crush=0 mode*/
  else  /* can's been crushed*/
  {
    crush=0;
    poke(0x7000,0x06); /*set plate upward*/
    while (analog(4) < 75)
      ;
    poke(0x7000,0x0);
    wait(15);
  }
  turn_off=0;
}
/*********************************************************/
```

```
void arbitrate()
{
  while(1)
  {
   if (deposit_can==0)      /*get can mode*/
   {
      if (((offir_left>95) || (offir_right>95)) || (follow1<500))
      {
        motor(0, fol_left);
        motor(1, fol_right);
        follow1 = follow1 + 1;
      }
      else
      {
        motor(0,0.0);
        motor(1,0.0);
        slide_base();
        while (analog(4)<100)
          ;
        test=3;
        crush_can();
        test=4;
      }
   } /*end get can mode*/
   else  /*deposit can mode*/
   {
```

```
if (following <= 500)
{
  if (((left_dif <=6) || (right_dif <= 6)) && ((onir_left > 95) || (onir_right
> 95) || (offir_left > 95) || (offir_right > 95)))
  {
    motor(0, fol_left);
    motor(1, fol_right);
    following = following + 1;
  }
  else if (((left_dif >= 6) || (right_dif >= 6)) && ((onir_left > 95) ||
(onir_right > 95) || (offir_left > 95) || (offir_right > 95)))
  {
    motor(0, obst_left);
    motor(1, obst_right);
  }
  else
  {
    motor(1, speed);
    motor(0, speed);
  }
} /*end following*/
else
{
  motor(0,0.0);
  motor(1,0.0);
  slide_base();
 }
```

```
    } /*end else deposit*/
  }  /*end while(1)*/
}
/*************************************************************/
void main()
{
  turn_off=0;
  deposit_can=0;
  crush=0;
  poke(0x7000,0x0);
  start_process(read_ir());
  start_process(avoid_obstacles());
  start_process(follow());
  start_process(arbitrate());
}
/*************************************************************/
```

APPENDIX D

APPENDIX E

Calculations For Can Crushing Power Screw

$d := .5 \cdot in$

$N := \frac{10}{in}$ screw dimensions

$P := 50 \cdot lb$ force being applied by screw

$\mu_c := .15$

$\mu := .15$ coefficients of friction

$L := 1 \cdot in$

$d_c := 1 \cdot in$

$dp := d - \frac{.649519}{N}$

$dp = 0.011 \cdot m$

$Tc := \mu_c \cdot P \cdot \frac{d_c}{2}$

$Tc = 0.043 \cdot kg \cdot m$

$Tsd := P \cdot \frac{dp}{2} \cdot \left(\frac{\mu \cdot \pi \cdot dp - L \cdot \cos(14.5)}{\pi \cdot dp \cdot \cos(14.5) + \mu \cdot L} \right)$

$Tsd = -0.209 \cdot kg \cdot m$

$Td := Tc + Tsd$

----------> $Td = -14.424 \cdot lb \cdot in$ Torque needed to drive screw

APPENDIX F