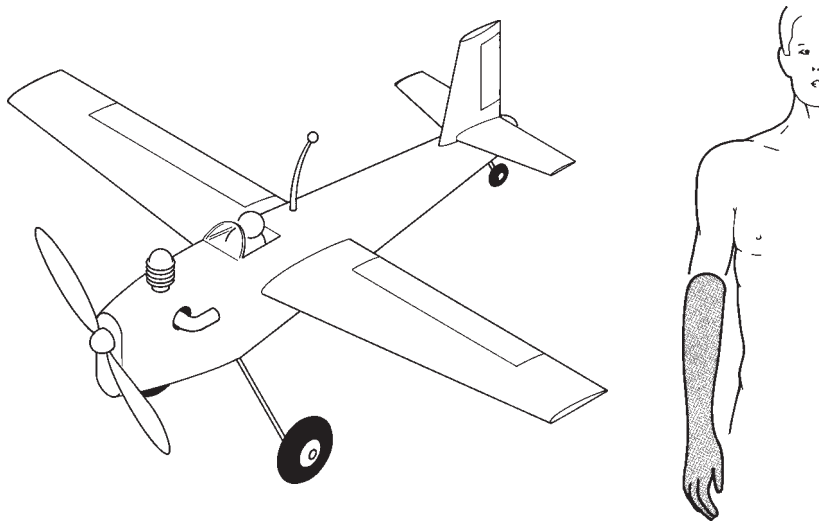
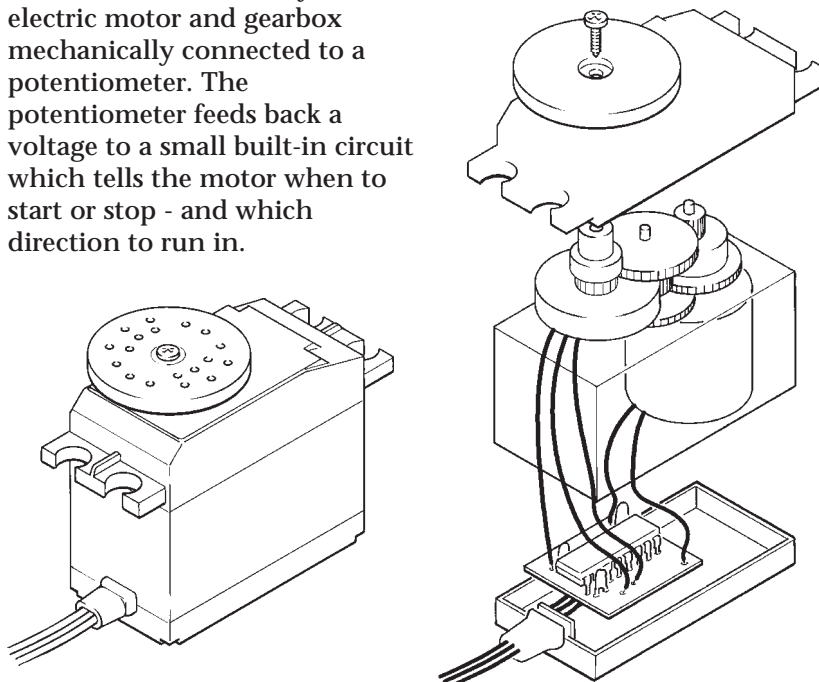


DESIGNING AND MAKING A SERVOMOTOR CONTROL SYSTEM

Systems involving servomotors are used extensively for providing very precise control movements. Widely differing examples include: radio control of models, actuation of robot arms, remote opening and closing of valves - and the driving and control of artificial limbs.



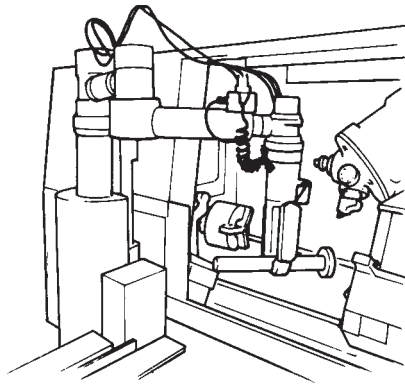
The servo motor systems used in advanced artificial limbs attempt to mimic the original limb which enables you, for example, to move your hand to a precise position, make it stop, grasp an object and place it elsewhere. In a servomotor system, movement is effected by an electric motor and gearbox mechanically connected to a potentiometer. The potentiometer feeds back a voltage to a small built-in circuit which tells the motor when to start or stop - and which direction to run in.



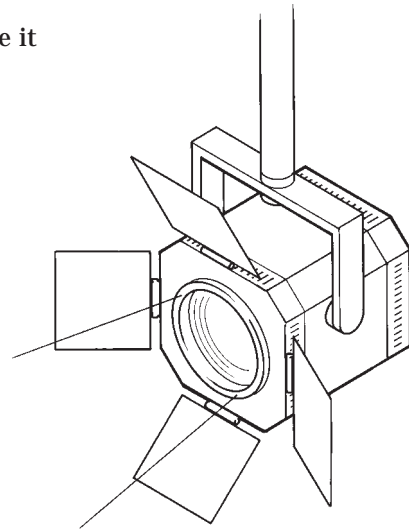
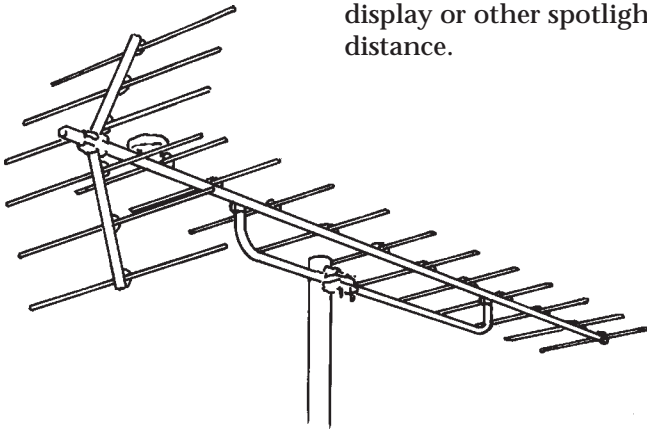
DESIGN OPPORTUNITIES

Computer controlled servomotor systems are used extensively in industry - for example in robots.

There are likely to be many local contexts where a servomotor can be used to effect precisely controlled movements. Examples include the following:

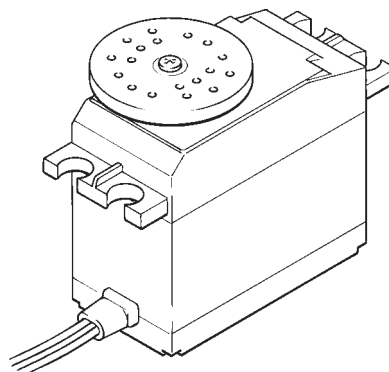


- A mechanism for rotating a TV aerial from a distance while it is being set up - i.e. the operator monitors the television picture for optimum reception.
- A remotely controlled robotic arm used for handling toxic materials in an enclosed and hermetically sealed space.
- A mechanism for positioning display or other spotlights from a distance.



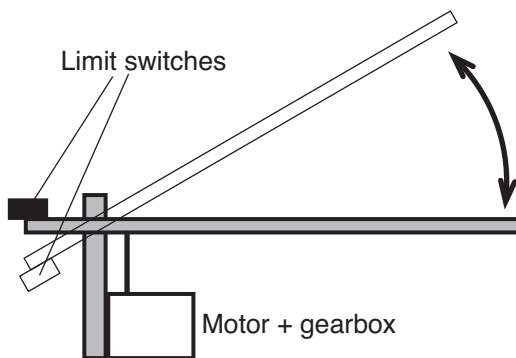
DESIGN SPECIFICATION

When a brief has been agreed on, it is necessary to draw up a design specification. This is a detailed description of what a system would be like, what it will do and who will use it. A servomotor system requires a servomotor unit and a means of providing it with a control signal. The following notes provide a commentary on control systems starting with a simple two-state system and concluding with a detailed description of how to drive servomotors. Because of the high cost of larger commercial servomotors, the example used throughout this text is the servo used in radio controlled models.



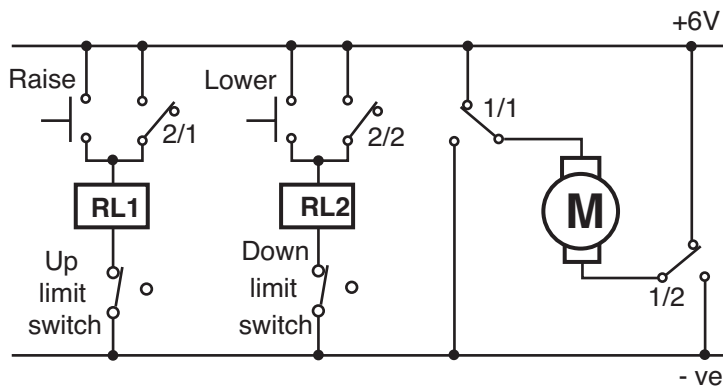
SIMPLE TWO STATE CONTROL SYSTEMS

These rely upon **limit switches**, or similar devices, to stop the motor when one of two conditions has been achieved, e.g. a car-park barrier. The motor is switched on to raise the barrier. When it is fully open it makes contact with a limit switch. This causes the motor to stop. The motor is reversed to lower the barrier. When it is fully closed it makes contact with a second limit switch. This causes the motor to stop again.



A car park barrier is a simple two-state control system

You can investigate this form of simple positional control by building a circuit using two relays.



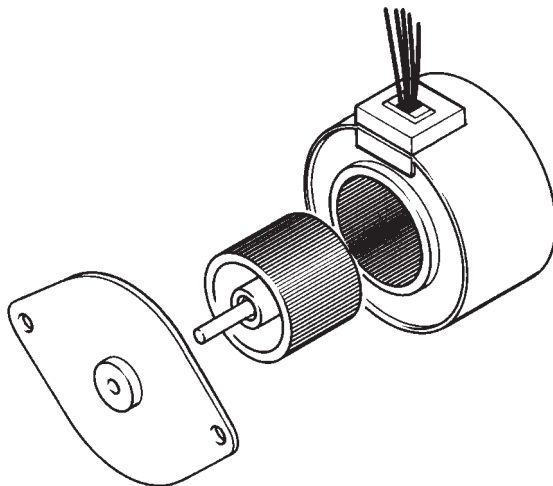
1/1 means contact 1 of relay 1
 2/1 means contact 2 of relay 2...etc.

Circuit operation - Imagine that the barrier is at the down position. All relay contacts will be at the position shown and the 'down' limit switch will be open. When the 'raise' button is pressed RL1 will energise. Contact 1/1 will switch over giving a current path to the motor which will allow it to turn. Contact 2/1 will switch over and latch RL1, causing the motor to continue turning after the 'raise' button is released. The motor will continue to turn until the barrier makes contact with the 'up' switch. When this happens the current path for RL1 will be broken and it will de-energise. Contact 1/1 will switch back and break the current path for the motor so it will stop. Contact 2/1 will switch back and unlatch RL1.

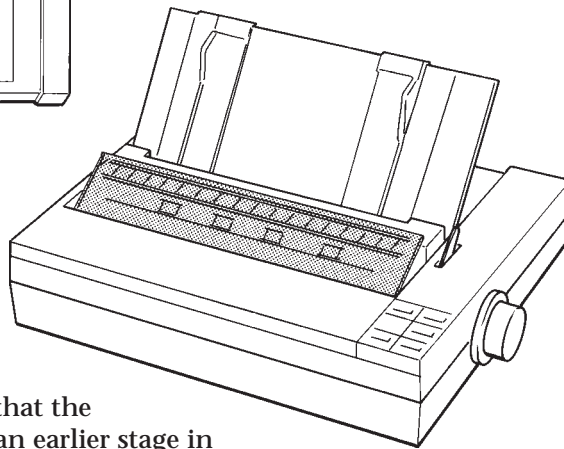
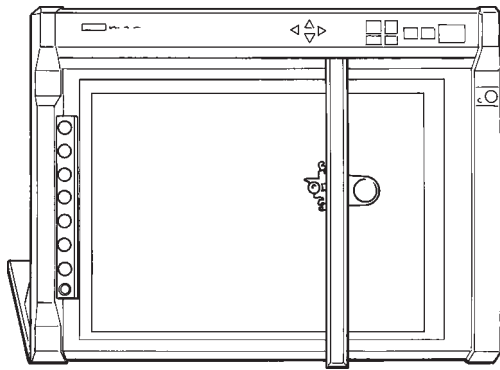
As the barrier is now in the up position the 'down' limit switch will be closed. When the 'lower' button is pressed RL2 will energise. Contact 1/2 will switch over giving a current path to the motor (note that the polarity has been reversed so the motor will turn in the opposite direction). Contact 2/2 will switch over and latch RL2, causing the motor to continue turning after the 'lower' button is released. The motor will continue to turn until the barrier makes contact with the 'down' switch. When this happens the current path for RL2 will be broken and it will de-energise. Contact 1/2 will switch back and break the current path for the motor so it will stop. Contact 2/2 will switch back and unlatch RL2.

A simple two-state control system can cope with this task as the barrier only ever has to be open or closed and never stop at a point that is somewhere in between. Designing control systems that can position something accurately **anywhere** between two points requires the use of more complex devices and circuitry.

A **stepper-motor** could be used to do this. These motors can be driven round in a series of small steps by a control circuit. If one step is equal to 7.5 degrees of angular rotation then driving the motor for a known number of steps would achieve a known number of degrees of rotation. E.g. $10 \text{ steps} \times 7.5^\circ = 75^\circ$ of rotation.

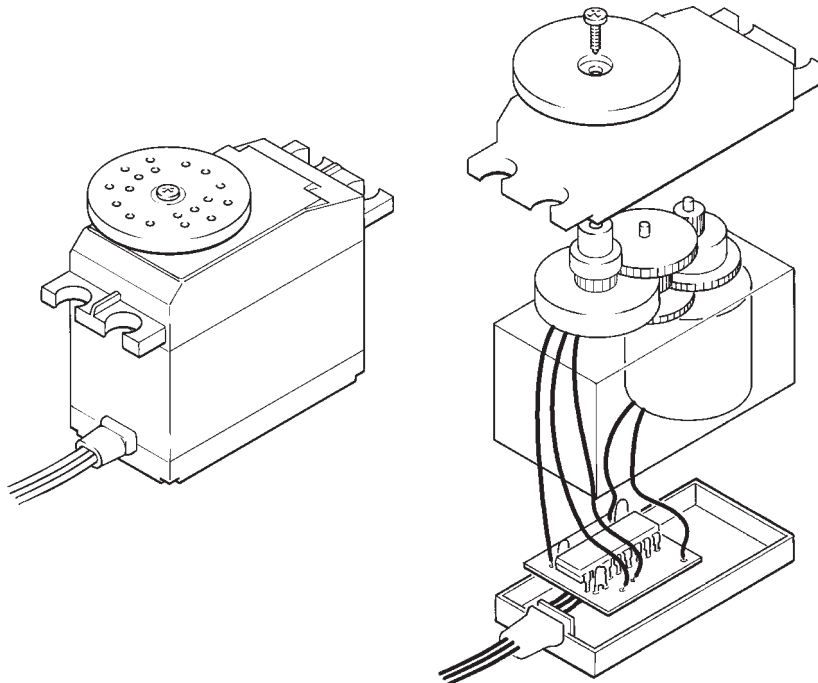


Stepper motors are a good solution for many positional control problems but there are some disadvantages. The main one is that the stepper motor can 'lose' its position. This happens if a very heavy load placed on the motor causes it to miss one or more steps. Stepper motors have quite a small torque or turning force. The problem is often overcome by checking the position of the stepper motor every so often. This may be done by using a limit switch at one extreme of the motor's movement. The motor is driven until the limit switch is operated and this point is used as a **Datum**. To check the position of the motor it is driven to the datum and then moved the desired number of steps away from it. To see this principle in operation you might look at a dot matrix printer or a plotter as these contain stepper motors. At the start of each operation the carriage or pen holder will drive to a datum position and then to the desired starting point.



Both the simple two state control system and the stepper-motor control system have **intermittent feedback**. This is provided by the limit switches. They 'feed back' the information that the system is at a known position, to an earlier stage in the control system. It is possible to design control systems that use **continuous feedback**. This removes the need to keep driving the motor to a datum point. Motor control systems that use this concept are known as **servomotors** (commonly referred to simply as servos).

Model control servos are readily available and are useful for experimenting, prototyping and modelling. They contain a motor, a gearbox and a control circuit.

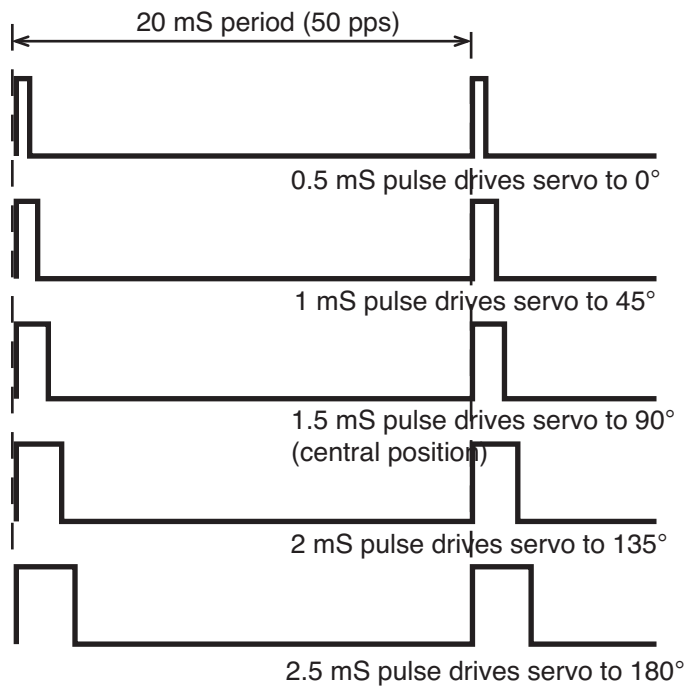


There are a number of different types of servos but they all operate in essentially the same way. The position of the motor is controlled by an internal circuit using continuous feedback from an internal potentiometer. There are two major types available, 90° and 180°. This refers to the amount of angular rotation that the servo can perform.

Typically, a servo has three connecting leads. Two of them are used for the power supply (5 volts). The positive connection is usually red and the negative black. The third wire is used to control the position of the servo. This wire is usually white.

Many servos are controlled by feeding a regular series of pulses to the control wire. The time between the pulses, or period, should be 20 mS (50 pulses per second). The position of the servo is controlled by the width of the pulse. A pulse width of 1.5 mS, for example, will move the servo to its central position.

Shortening the pulse will move the servo in one direction and lengthening it will move it in the opposite direction. Once the servo has reached a position it will stay there as long as the pulse width stays the same. If the pulse width alters then the servo will take up the new position. This method of control is known as **pulse duration modulation** (p.d.m.).



INVESTIGATING A 180° P.D.M. MODEL CONTROL SERVO

Supplying a regular series of pulses to a servo with the facility to change the duration of the pulses requires the use of some electronic circuitry. This unit will examine two methods of providing this:

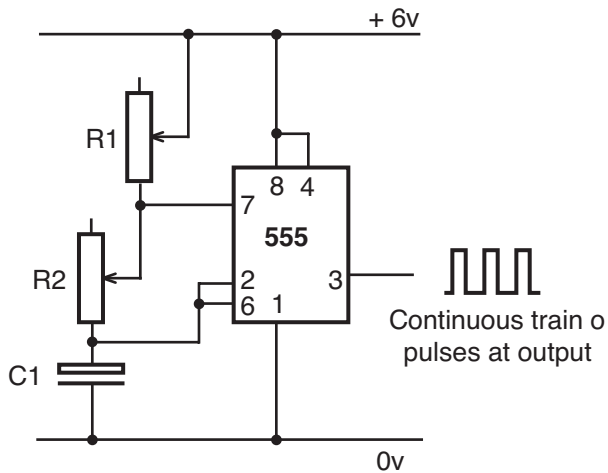
- how to use two 555 ICs to produce the pulse train to control a single servo,
- how the TEP PLC Chip can be used to control a number of servos using different input devices.

The p.d.m. servo can be used as the basis for many positional control projects.

Controlling a P.D.M. Servo using 555 ICs

The 555 is a general purpose timer I.C. that can be used in one of two modes, either **astable** or **monostable**.

Astable mode - The device will supply a continuous train of pulses at its output. The frequency and duration of the pulses is set by a network of two resistors and one capacitor (R_1, R_2, C_1).

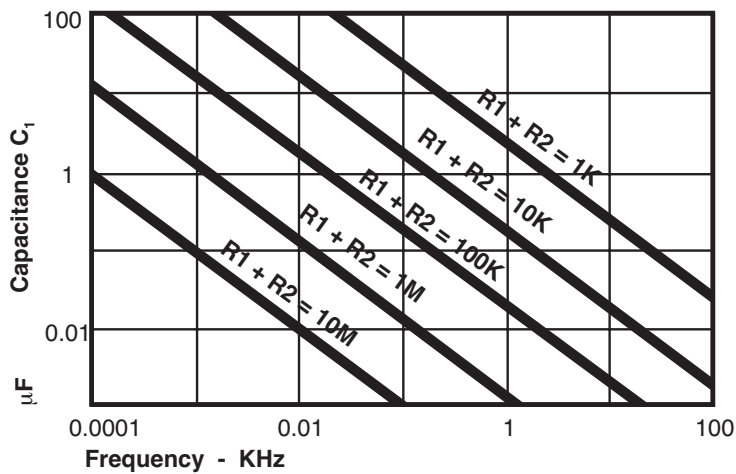


It is possible to calculate the frequency and duration of the pulses using the following formulas:

$$\text{pulse duration} = 0.693 (R_1 + R_2)C_1$$

$$\text{frequency} = \frac{1}{(R_1 + 2R_2)C_1}$$

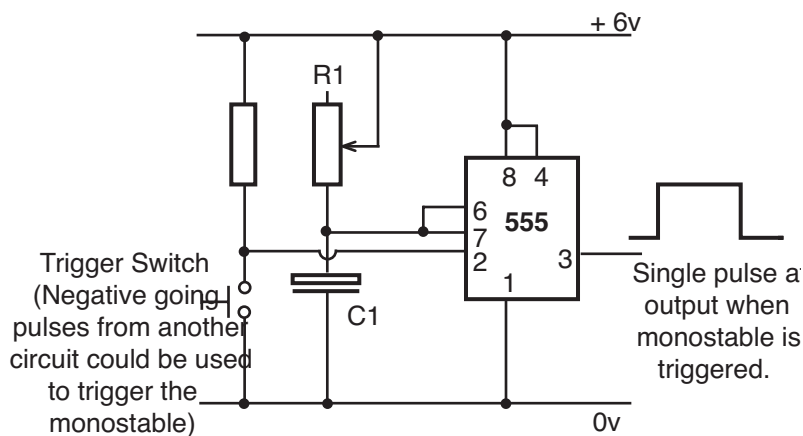
It can be easier, though, to use a table, or series chart, to look up the **very approximate** starting values for R_1 , R_2 and C_1 .



Simply select the desired frequency on the horizontal axis, draw a line from this point up the graph until you intersect with one of the diagonal resistance lines. This gives you the value for $R_1 + R_2$.

Then draw a horizontal line across from this intersection point to the capacitance values on the vertical axis. This gives you a value for C_1 .

Monostable mode - The output from the device will switch from low to high when it is triggered. (Low = 0 volts, high = supply voltage.) The output will stay high for an amount of time that is set by the resistor capacitor network of R_1 and C_1 . Once this time has passed the output will go low again. It will stay low until the device is triggered again.

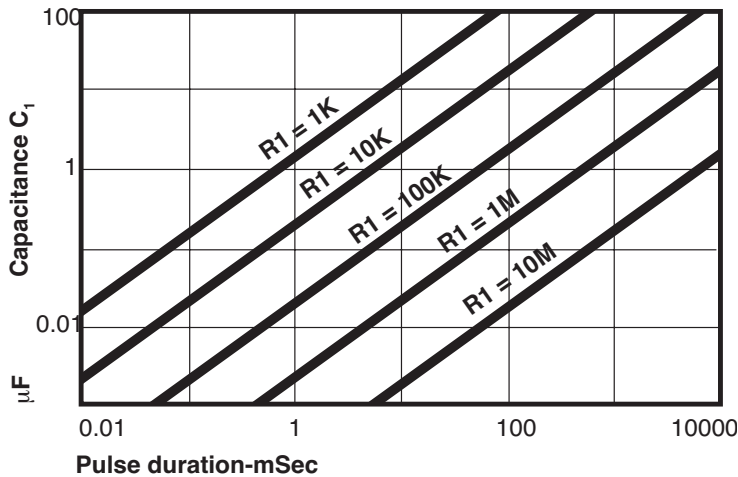


The monostable is triggered by pulling pin 2 down from the supply voltage to 0 volts. This can be done using a switch or by another control circuit.

The duration of the output pulse can be calculated by using the formula:

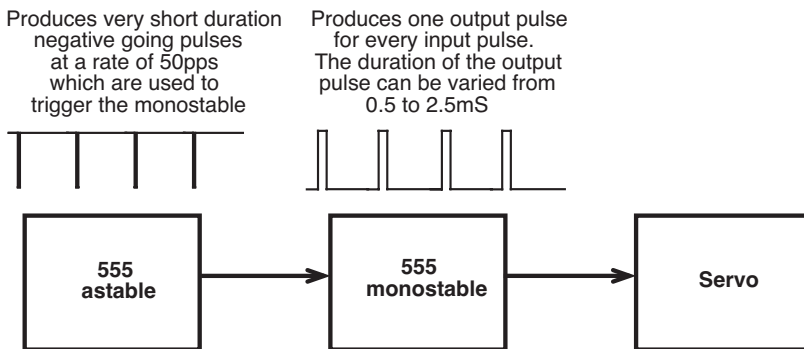
$$\text{Pulse duration} = 1.1 R_1 C_1$$

Again you can use a series chart to find the values of R_1 and C_1 .



Select the desired pulse duration from the horizontal axis. Draw a line up the graph until you intersect with one of the diagonal resistance lines. This gives you the value for R_1 . Draw a line across from the intersection point towards the capacitance values on the vertical axis. This gives you the value for C_1 .

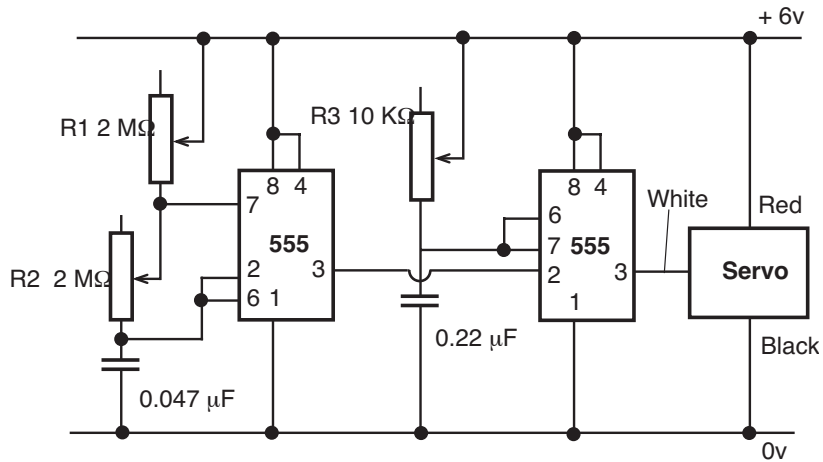
A simple servo controller can be built using two 555 ICs. The first is connected as an astable. The pulses produced by it are used to trigger the second device that is connected as a monostable.



P.D.M. servo controller block diagram

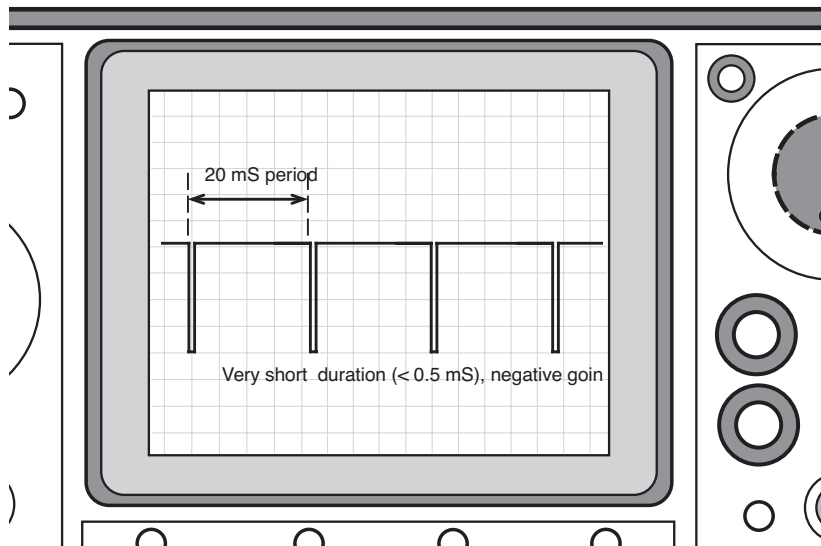
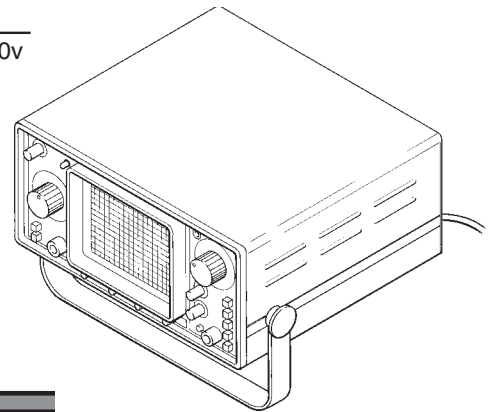
The first stage sets the frequency of the pulse train (20 mS) and the second stage allows the pulse duration to be varied (0.5 to 2.5 mS).

This is the complete circuit diagram for the servo controller with all component values.



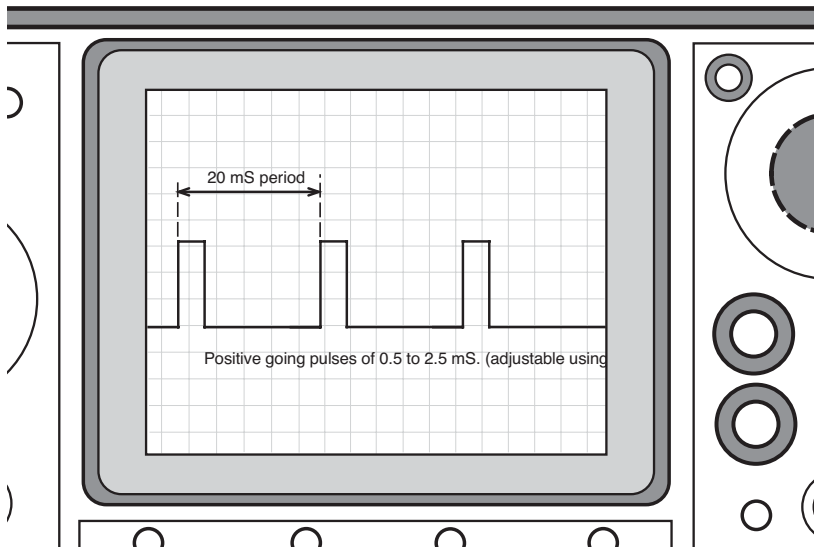
Once you have built the circuit you will need to set it up. To do this you will need to use an oscilloscope.

First connect the oscilloscope to the output of the astable (pin 3). You need to adjust R_1 and R_2 until the trace looks like this.



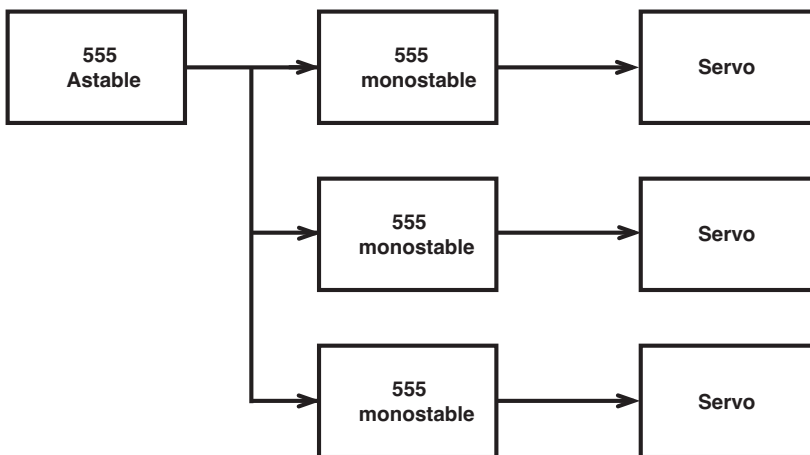
The gap between the leading edges of each negative going pulse should be 20 mS. The duration of each pulse should be very short. The actual duration is not important as this will be set by the next stage, but it must be less than 0.5 mS. The negative going pulses are used to trigger the monostable.

Next connect the oscilloscope to the output of the monostable and check that you have a trace like this.



You should be able to change the duration of the positive going output pulses by adjusting R_3 . They should vary in duration from 0.5 to 2.5 mS. The gap between the leading edges should remain constant at 20 mS.

If you want to control more than one servo then it is not necessary to duplicate the entire control circuit. You can simply add on extra monostable circuits.



Block diagram for a multiple servo controller

This basic circuit is a simple solution to quite a complex control problem. The majority of the work is done for you inside the servo!

CONTROLLING A P.D.M. SERVO USING THE TEP PLC CHIP

(See Technology Study File 3 for basic information on the TEP PLC chip.)

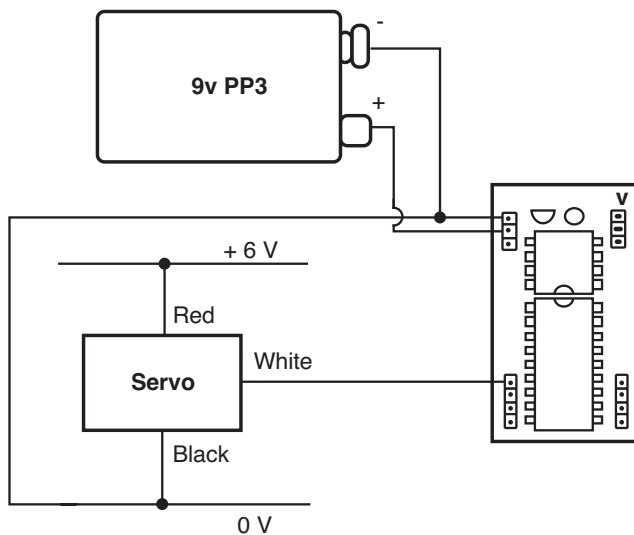
Using the TEP PLC Chip to control a servo is simplicity itself. The control signal is provided using the **pulsout** command.

pulsout *pin,duration*

Pin - i/o pin to use

duration - pulse width in units of 10 μ S

Connect the PLC Chip to the servo like this.



Note the use of a separate 6 volt supply to power the and that both negatives are connected.

Down-load this program

```
servo:  pulsout 0,150
        pause 20
        goto servo
```

The servo should drive to its centre position.

This is because the output pulse is 150 \times 10 μ S (1.5 mS) long. The 'pause 20' in the program gives a 20 mS gap between pulses.

Now try modifying the program like this

```
servo:  pulsout 0,200
        pause 20
        goto servo
```

When the program is down-loaded the pulse width will have increased to 2 mS. The servo should drive to 90 $^\circ$.

Now try:

```
servo:  pulsout 0,100
        pause 20
        goto servo
```

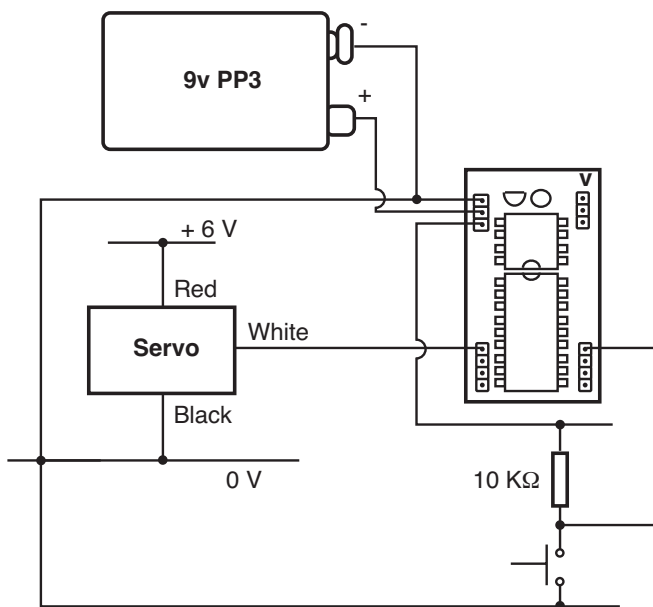
When the program is down-loaded the pulse width will have decreased to 1 mS. The servo should drive to 0°

N.B.

If you have a 90° servo then you can use durations of 100 to 200.
 If you have a 180° servo then you can use duration of 50 to 250.
 A duration of 150 will be the centre position for both devices.

Next you need to add an input device to control the position of the servo. You can use any of the input devices that are shown in the study file that produce a logic 1 or logic 0.

This example use a simple push switch. Connect the switch like this.



Now down-load this program.

```
servo:  input 7
        if pin7 = 0 then left
        if pin7 = 1 then right
```

```
left:  pulsout 0,100
        pause 20
        goto servo
```

```
right: pulsout 0,200
        pause 20
        goto servo
```

When the switch is pressed and held down the servo should drive to 0°. When the switch is released the servo should drive to 90°.

How it works

servo:

input 7 (**sets pin 7 to an input**)

if pin7 = 0 then left

(if input is a 0 then the program branches to the left procedure)

if pin7 = 1 then right

(if input is a 1 then the program branches to the right procedure)

left:

pulsout 0,100

pause 20

(outputs a 1 mS pulse followed by a 20 mS space)

goto servo

(returns the program to the start to check the state of the switch)

right:

pulsout 0,200

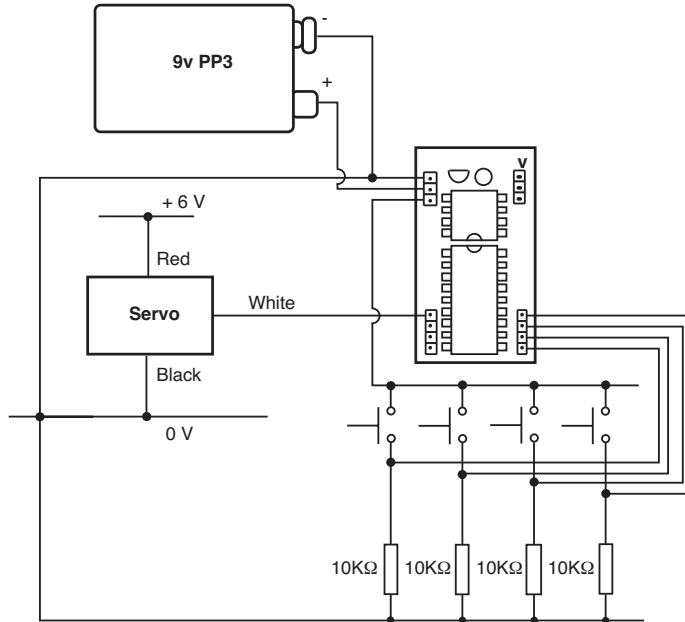
pause 20

(outputs a 2 mS pulse followed by a 20 mS space)

goto servo

(returns the program to the start to check the state of the switch)

This program will only allow you to set the servo to one of two positions. You could add more switches to increase to number of points that the servo drives to, e.g. connect the extra switches like this.



Now down-load this program.

```

servo:  input 4
        input 5
        input 6
        input 7
        if pin4 = 1 then hleft
        if pin5 = 1 then left
        if pin6 = 1 then right
        if pin7 = 1 then hright
        pulsout 0,150
        pause 20
        goto servo

hleft:  pulsout 0,100
        pause 20
        goto servo

left:   pulsout 0,125
        pause 20
        goto servo

right:  pulsout 0,175
        pause 20
        goto servo

hright: pulsout 0,200
        pause 20
        goto servo
    
```

If none of the switches are pressed then the program stays in the endless loop labelled 'servo'. The servo drives to its centre position.

If any of the switches are pressed then the program branches into the appropriate sub-routine, drives the servo to the new position then returns to 'servo' to check the state of the switches.

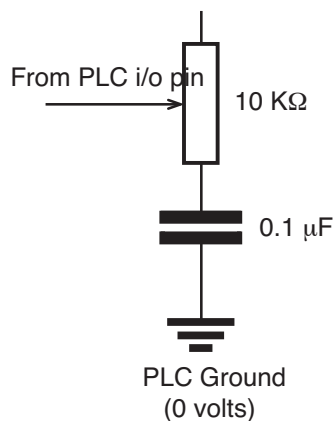
Pressing switch 4 should drive the servo hard left.

Pressing switch 5 should drive the servo mid-left.

Pressing switch 6 should drive the servo mid-right.

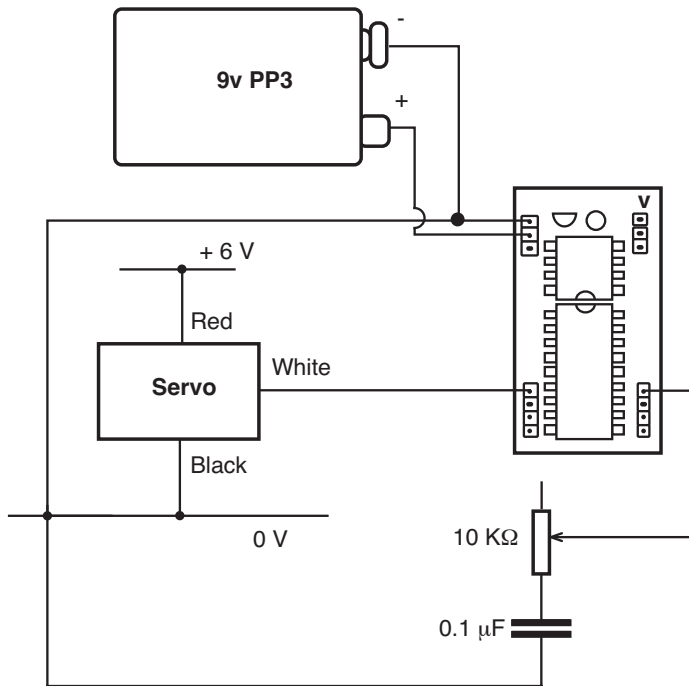
Pressing switch 7 should drive the servo hard right.

Although this improves the level of control on the servo it is still not possible to drive it to any position in its range and hold it there. To do this you need to use a potentiometer (pot) as the input device. It is possible to program the PLC Chip to read the value of the potentiometer, store this value in the memory and then use it to control the position of the servo. To do this you must connect the pot to a capacitor like this.



The PLC Chip reads the value of the pot by sensing how long it takes to charge the capacitor. A long time indicates that the pot is adjusted to a high value and a short time indicates that the pot is adjusted to a low value.

Connect the PLC Chip, the servo and the pot like this.



Now you have to do a test on the pot to get the PLC Chip to recognise it. Do this by pressing

Alt P

A program is automatically down-loaded to the PLC Chip to check the size and range of the pot. You should see a box appear on the screen.

Type in the number of the pin that the pot is connected to. In this case 7.

The figure 7 in the top of the box should be highlighted. Now press enter.

The box should change. On the left will be **scale** and on the right will be **read**.

Now adjust the pot. The number next to scale should change. Adjust the pot until you get the minimum value for scale. **Note down this number as you will use it later.**

When you have done this press the space-bar. This should lock the scale and read the value of the pot. The number next to read should vary between 0 and 255 as you adjust the pot.

To get rid of the pot test box press **esc**.

Now you need to write the program. To use the pot in a procedure you use the command,

pot pin,scale,location

pin - the i/o pin that the pot is connected to.

scale - the number that you noted down when you tested the pot.

location - the memory address that the value will be stored in (b1 to b13).

Down-load this program (don't type 'scale' in the first line. Use the number that you noted down when you tested the pot),

```
servo:  pot 7,scale,b1
        pulsout 0,b1
        pause 20
        goto servo
```

When you adjust the pot the servo should move. The position of the pot and the servo should roughly match up.

How it works

servo:

pot 7,**scale**,b1

(reads the value of the pot connected to pin 7 and stores the value in memory address b1)

pulsout 0,b1

(outputs a pulse on pin 0 that has a duration of the value stored in b1x10µS)

pause 20

(waits for 20 mS to give the correct output frequency)

goto servo

(returns the program to the start to check the value of the pot again)

This simple four line program is not perfect and you may have noticed problems when you adjust the pot to the very ends of its travel. This is because the output pulse will be outside the limits acceptable to the servo. This is from 5 to 2.5 mS for a 180° servo. The width of the pulse is set by the value that is stored in b1. This can vary between 0 and 255. You can overcome this problem by putting limits on the value of b1 like this,

```

servo:  pot 7,scale,b1
        if b1 < 50 then toolow
        if b1 > 250 then toohigh
        pulsout 0,b1
        pause 20
        goto servo

toolow: pulsout 0,50
        pause 20
        goto servo

toohigh: pulsout 0,250
        pause 20
        goto servo

```

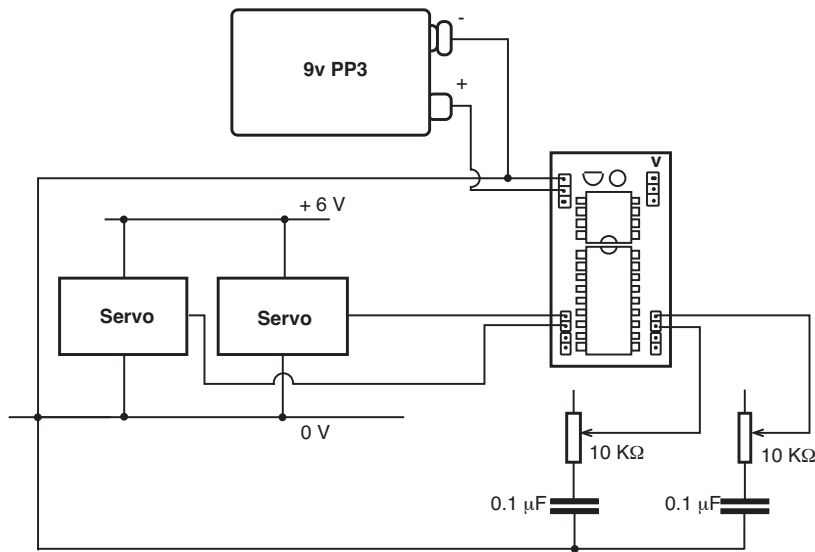
When you down-load this program the problem should not occur. If the value of b1 is too low the program branches and outputs a pulse of 5 mS in the 'toolow' procedure. If the value of b1 is too high then the program branches and outputs a pulse of 2.5 mS in the 'toohigh' procedure.

N.B. These figures are for a 180° servo; you would need to adjust them to 100 and 200 for a 90° servo

It is possible to use the PLC Chip to control more than one servo. It takes one i/o line for the input device and one for the servo, hence, it is possible to control up to four servos at one time.

You will of course have to modify the program. This example examines the control of two servos.

Connect the servos, the pots and the PLC Chip like this.



Now you must check the scale value for each pot as before and note down the scale value.

The program will need to check the value for each pot, store this in memory and then output a pulse of the correct duration to each servo. Try this,

```

servo:  pot 7,scale,b1
        pot 6,scale,b2
        pulsout 0,b1
        pulsout 1,b2
        pause 20
        goto servo
    
```

when the program is downloaded the pot connected to pin 7 should control the servo connected to pin 0 and the pot connected to pin 6 should control the servo connected to pin 1.

Again you may notice problems when the pots are at the ends of their scales. This can be overcome in a similar way as before by limiting the minimum and maximum values for b1 and b2.

Try this program,

```

servo1:  pot 7,scale,b1
         if b1 > 250 then th1
         if b1 < 50 then tl1
         pulsout 0,b1
         goto servo2

th1:     pulsout 0,250
         goto servo2

tl1:     pulsout 0,50
         goto servo2

servo2:  pot 6,scale,b2
         if b2 >250 then th2
         if b2 < 50 then tl2
         pulsout 1,b2
         pause 20
         goto servo1

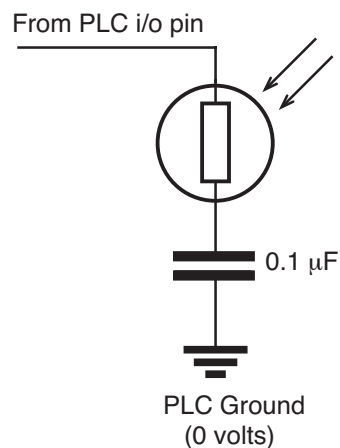
th2:     pulsout 1,250
         pause 20
         goto servo1

tl2:     pulsout 1,50
         pause 20
         goto servo1
    
```

This should overcome the problem. If the values of b1 and b2 stay within limits then the only operational procedures in the program are 'servo1' and 'servo2'. If the values step outside the limits then th1,th2,tl1 and tl2 come into play and limit them.

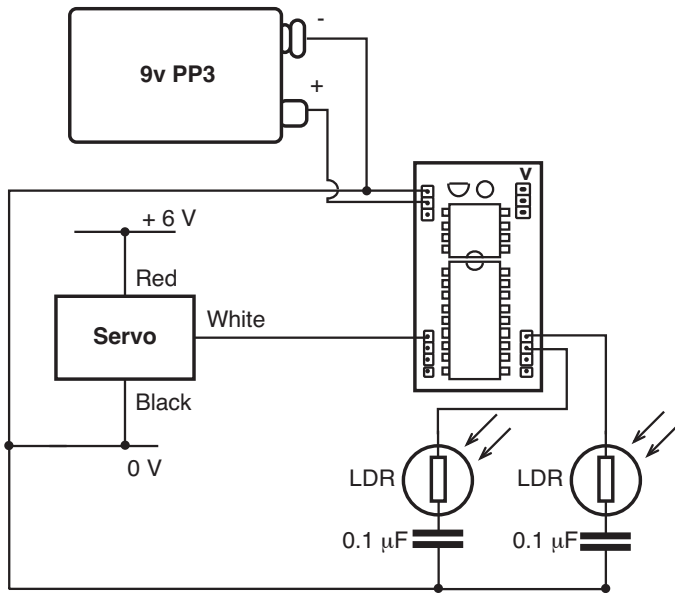
N.B. The figures used are for a 180° servo you would need to change them to 100 and 200 for a 90° servo.

It is possible to use other resistive devices in place of the pot. A standard ORP 12 LDR can be connected to the PLC Chip.

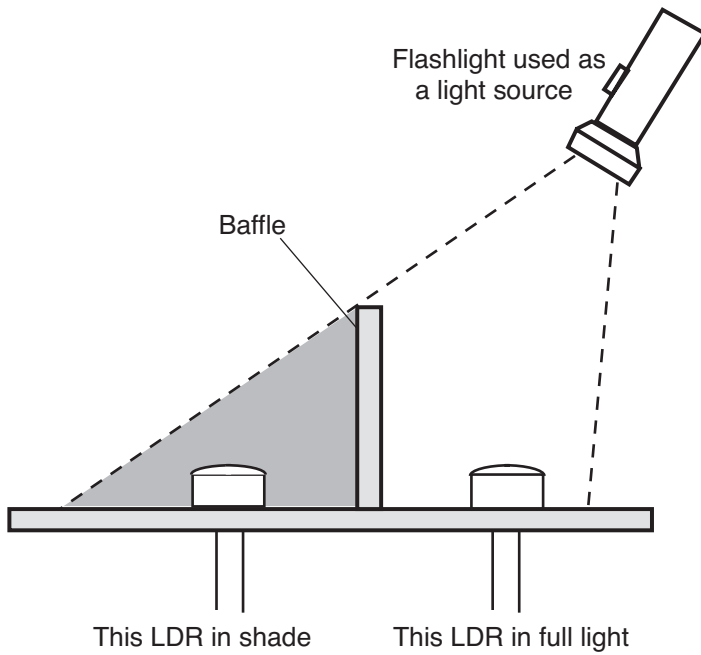


As the light level changes then the value of the number stored will change also. This example uses two LDRs to control 1 servo. It will cause the servo to drive towards a source of light.

Connect the LDRs, the servo and the PLC Chip.



It is a good idea to separate the two LDRs by some form of baffle.



Next you need to scale the LDRs in the same way as for the pots. You cannot adjust them so you will have to shine the light source on to them and then take it away. Press the space-bar when you have the smallest value for **scale**. (If the scale value stays at 255 then don't worry; you may still be able to read a change on the LDR)

The program will need to read the values from each LDR and store them in memory. If they are equal then the servo should be in its central position. If one is higher than the other then the servo should drive towards the lowest resistance/highest light level. This is not too difficult but requires some maths in the program. Try this:

```

servo:  pot 7,scale,b1
        pot 6,scale,b2
        b3 = 150+b1-b2
        pulsout 0,b3
        pause 20
        goto servo
    
```

The output pulse is controlled by the value in b3. If the light is shining equally on both LDRs then b1 and b2 must be equal. The servo will drive to its centre position because b3 will equal 150. If the light source moves then the values of b1 and b2 will change. One will rise and one will fall. The line 'b3=150+b1-b2' will adjust the value of b3 to be either less than or greater than 150, driving the servo towards the light source.

There is scope for much further experimentation using the PLC Chip to control servos. As long as you follow the basic steps that are outlined above you should achieve success.

