Lab 4

Servo Motors and Gears

Authors:

Alexandre Mesot, Jonas Lussi, Naveen Shamsudhin and Prof. Bradley J. Nelson

Multi-Scale Robotics Lab, Institute of Robotics and Intelligent Systems

Date: 2020

Version: 1.0

Summary: In this lab, we will introduce servo motors and gears by solving some simple taks. We will learn:

- How servo motors work and how to control them using the Arduino interface.
- How to increase/decrease the torque and run length of the servo motors using combinations of gears.
- How to design gear trains for specific applications and requirements

4.1 Background

There are many different types and divisions of electric motors: AC or DC motors, Stepper Motors, Rotary and Linear motors and many more. Servo motors (or servos in short) are linear or rotational motors with a very high positional precision. Any applications that need to move or rotate an object to a precise location will use servo motors. Such applications are manifold and range from RC planes and toy cars, to opening and closing DVD and Blu-ray Disc trays to operating grippers in all kinds of robots. Servos are also often used for in-line manufacturing, where they shine in highly repetitive, yet very precise work areas. Servos are generally fairly small and offer a very high torque compared to their size. The IRM lecture notes on **motors** provide additional background information for this lab and feel free to do your own research if your interest has been piqued.

4.1.1 Servos, control and pulse width modulation (PWM)

In the housing of servos there are 4 components: A (DC or AC) motor, a gear assembly, a potentiometer, and a controlling circuit (Figure 4.1). The gear assembly reduces the rotational speed of the motor in order to increase its torque. The potentiometer is attached to the drive shaft and is used to determine the position of the servo. The control circuit reads the output of the potentiometer and compares it to the the command signal, which it receives from the user. The difference is processed in a feedback loop which outputs a signal for the motor to start rotating. This will start turning the potentiometer as well, thus reducing the difference between its output and the command signal. The loop is repeated until the difference is zero and the new position has been reached . This position measuring mechanism is the reason why servos usually can only rotate 180°(or any fixed angle). While the DC motor can rotate continuously, a potentiometer cannot. When the difference in position is large the servo will move as fast as possible to close the gap. As it gets closer to its desired position it will continually slow down so as to not overshoot the position.

Potentiometer: A Potentiometer is a three-terminal variable voltage divider. It has a resistive element inside. At each end of the element one terminal is rigidly attached. The third, central terminal wipes over the resistive element when the knob is turned. The closer the wiper is to the end terminal, the less the resistance is. The further away, the greater it becomes.

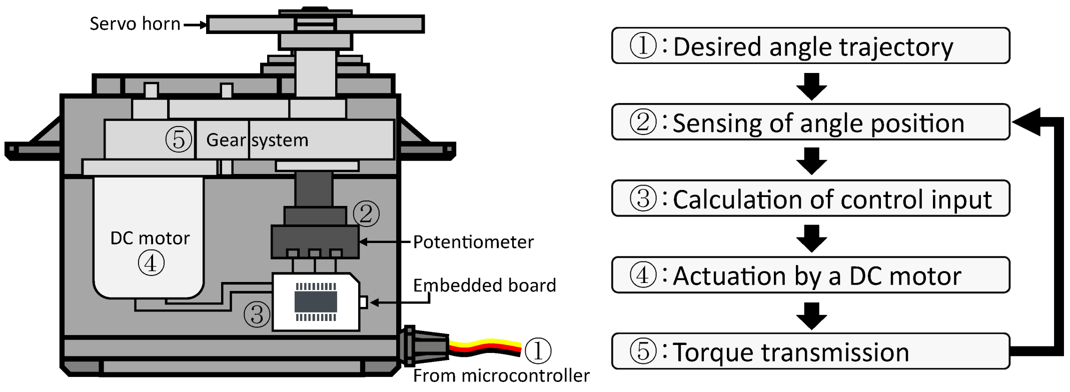
Servo horn
⑤ Gear system
DC motor ④
② Potentiometer
Embedded board
③
① From microcontroller
①: Desired angle trajectory
②: Sensing of angle position
③: Calculation of control input
④: Actuation by a DC motor
⑤: Torque transmission

Figure 4.1: Schematic of a DC servo motor and sequence to move to a desired position.

Servos can usually rotate from 0°to 180°. The command signal to set a desired position is sent in the form of a PWM (Pulse Width Modulation) signal. This means that the position of the servo is encoded in the length of the pulse that is sent. Generally servo motors expect to receive a pulse every 20 ms, with the pulse width usually ranging from 1 ms (to encode the position of 0°) to 2 ms (to encode the position of 180°). The position scales linearly between the two extremes, meaning that to reach the 90°position a PWM of 1.5 ms is needed (Figure 4.2). Keep in mind that the servo needs some time to reach its position. Some servos come with an additional cable, which outputs its current position. This can be used in a feedback loop to perform a series of consecutive movements without having to estimate how long it will take for the servo to reach each position.

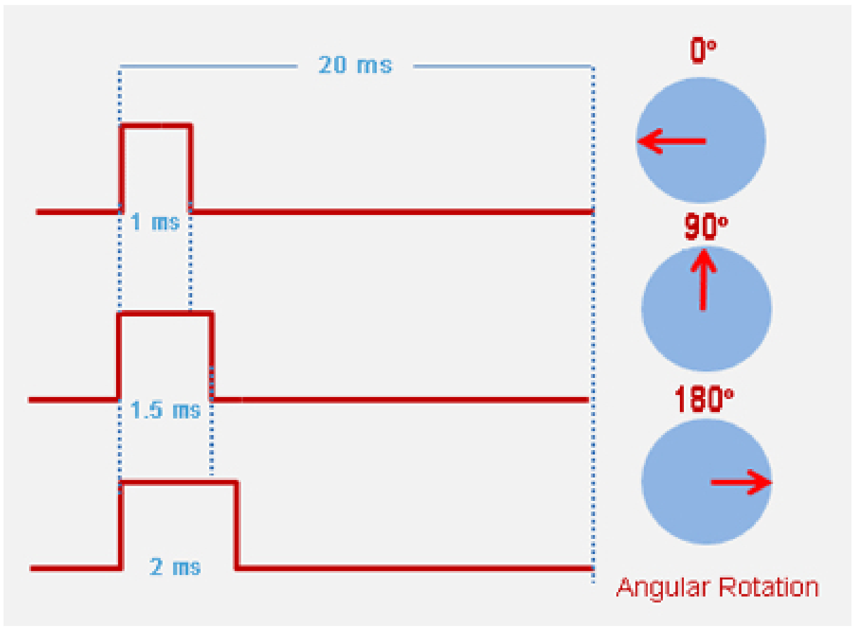

Figure 4.2: Pulse Width Modulation is used to encode the position of the servo in the length of a pulse.

Note that continuous rotation servo motors which can rotate indefinitely in any direction also exist. Instead of having a very precise control over its position, continuous rotation servos have a very precise control over speed and direction. The rotational speed is determined using a similar principle as with the normal servos. The potentiometer is replaced by a pair of identical resistors that create a fixed voltage divider which acts as if the potentiometer is always at its mid point (let's say 90°). When you send a PWM signal for the 180° position the servo will start moving fast, in order to reduce the large gap. However as there is no potentiometer the gap will remain the same and the speed will remain constant. If you now send a PWM signal for the 85° position the servo will start turning in the other direction, as well as turning very slowly, as the gap is much smaller than before. This way the control circuit can be the same in both types of servos and the only thing that has to be changed is to replace the potentiometer with a pair of resistors.

4.1.2 Torque and Gears

As you have seen in the lecture, electric motors have a linear relationship between rotation speed and torque. The operational rotation speed at which the output power is maximal is at $\omega = 0.5 * \omega\_max$. Typical rotation speeds in motors are in the 1000s of rpm, in order to maintain this ideal power output. Applications however need more reasonable rotation speeds, as nobody wants to run through a rotating door at 30 turns per second. Combinations of gearheads are thus very important in robotics and almost any kind of motor applications because they reduce the rpm and at the same time increase the load that the motors can handle. This can be seen by looking at the relationship between power, rpm and torque: $P = \omega * T$. The ideal power output of the motor is kept constant. By using gear trains the rotational speed is increased which then leads to an increased torque. It is the same principle that is used in bicycles and cars when gears are shifted in order to maintain the engine (or your legs) in the regime where most power can be generated.

In this lab you will be working with spur gearheads in order to increase the torque of a servo motor, as well as to increase its range of motion. Real life applications have additional requirements that will make the choice of the correct gears more elaborate. Accuracy, backlash, efficiency, lifetime, load, noise reduction, inertia matching and torque multiplication often have all to be considered in the decision-making process. Here are some important relationships:

Gear Ratio: The gear ratio is the ratio between the rotational speeds of two connecting gears. As each gear has a different diameter, each axis rotates at a different speed when engaged. The gear ratio is calculated by dividing the output speed by the input speed ($r = \omega_{in}/\omega_{out}$), or by dividing the number of teeth of the driven gear, by the number of teeth of the driving gear ($r = Z_{driven}/Z_{drive}$).

Torque: The amount of torque that is transmitted is also proportional to the gear ratio. The output torque is calculated by dividing the input torque by the gear ratio ($T_{out} = T_{in}/r$)

Compound gears: When two or more gears are fixed together on the same axis of rotation they are called a compound gear (Figure 4.3). Compound gears rotate at the same speed, even though they usually have a different amount of teeth. This allows for a much greater increase or decrease of speed from the driving to the driven shaft, compared to a classical gear train. The gear ratio for the whole compound gear train, assuming gear A is the driving gear, is: $r = Z_{GearB}/Z_{GearA} * Z_{GearD}/Z_{GearC}$. Follow this link if you want more information about compound gears:

http://www.technologystudent.com/gears1/gears3.htm

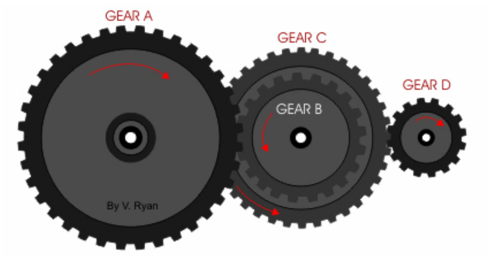

Figure 4.3: Compound gear.

4.1.3 Adafruit Motor Shield

Arduino boards are very useful for general electronic applications, however some specific applications need additional features. Shields are pieces of hardware that you can mount on the Arduino in order to add these additional features. Popular examples are Ethernet Shields to add internet capabilities, GSM/GPRS Shields to add mobile data connectivity, LCD Shields to add an external display and Bluetooth Shields to connect the Arduino with your phone. In the case of robotics, the Arduino board quickly reaches its limit, as its pins cannot supply enough power to drive multiple motors. Motor shields can support that additional load as they have their individual power supplies, without altering the simple and useful Arduino interface. If no power supply is attached to the Motor shield the power comes directly from the 5V power output of the Arduino. The maximum current that the Arduino board can supply is 800 mA, which is enough to drive a single Servo. It is highly recommended to connect a 5 V or 6 V power supply to the motor shield, if additional Servo motors are added, as you might fry the Arduino board. Some motor shields can be stacked to theoretically drive hundreds of servos and motors. The motor shield that we will be using in this lab is an Adafruit Motor Shield V2 with an optional power supply of 5V or 6V and specific connectors for servo motors, as well as stepper and DC motors. As you can see in Figure 4.4 the motor shield is plugged directly into the pins of the Arduino board. All of the pins are redirected to the top of the motor shield, where they can be used normally. Additionally GPIO pins 9 and 10 are redirected and each aligned with a ground and a power pin, so that servo motors can be plugged in without having to separate the servo cables (Red arrow in Figure 4.4).

Check out the following link if you want to know more about the Adafruit Motor Shield:

https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino

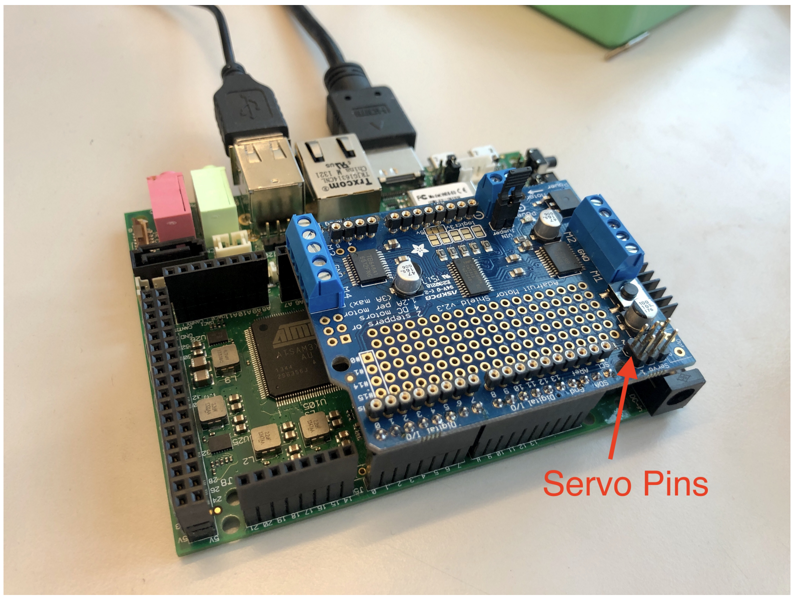

Figure 4.4: Udoo with the Adafruit Motor Shield V2.3 plugged in. The arrow shows the 2 Servo pins.

4.1.4 Polling vs Interrupts

If you need your Arduino code to wait for an input in order to do something (for example press a button to turn on an LED) it has two main ways to check for the input. Polling and Interrupts. Polling is used when you check for the input every time the main loop is run. Interrupts run in the background, checking for input and freeing the main loop for code that is not related to the input. The Interrupt Service Routine (ISR) will interrupt the main code, as soon as the input is received. Here is an analogy:

You are expecting a message from somebody. If you pause what you are doing every minute to check your phone in order to make sure that you didn't miss it, it's called polling. If you set a vibration or ring tone, that alerts you when the message arrives, it's called an interrupt. Now you can continuously do whatever you were doing undisturbed, until you get the message. Only then do you interrupt your workflow and look at the phone.

Polling is fine for small, uncomplicated code. However if you have multiple different tasks running it can be a valuable saving of computing power and time to use interrupts. Figure 4.5 illustrates this well.

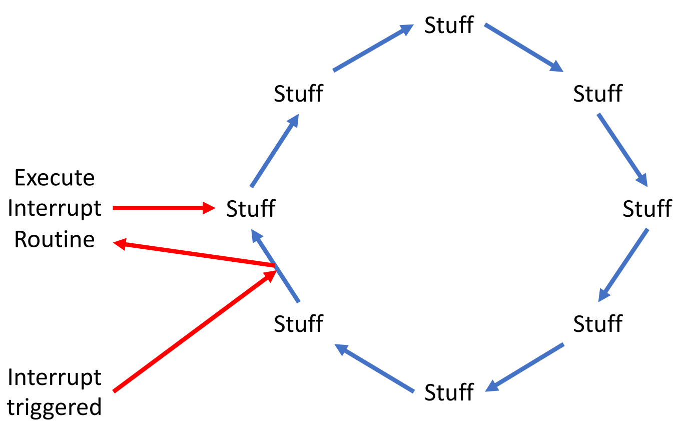

Figure 4.5: The main loop is only interrupted if the ISR is activated by an input.

If you want more information on how to implement interrupts on Arduino boards check out **this** link.

4.2 Prelab Procedure

Note: Prelab assignments must be done before reporting to the lab and must be turned in to the lab instructor at the beginning of your lab session. Additionally, you also have to upload your solution as a single PDF-file to moodle. Make sure to upload the file before your lab session starts, late submissions will not be corrected. The prelab needs to be handed in as a group. All the prelab tasks are marked with **PreLab Qx**.

In this lab we will be using the Futuba s3003, which is a common DC servo motor. It has 3 cables connected to it: Black (ground), Red (5V-6V power) and White (Signal). Read the background on servos and gears as well as the data sheet of the Futuba s3003 and answer the following questions:

1. What is the range of PWM pulse lengths that can be used to set the position of the Futuba s3003 servo? **(PreLab Q1)**
2. What PWM pulse length do you need to set to position the servo at 66.6°? **(PreLab Q2)**
3. What is the fastest that the servo can run through its full, 180° range of motion? **(PreLab Q3)**
4. Is this s3003 servo a good or a bad choice for a clock? Explain. **(PreLab Q4)**
5. List a different application that you think could use the high position accuracy of servos (that has not yet been listed in the introduciton).**(PreLab Q5)**
6. What is the result of increasing the input voltage of the servo from 5V to 6V in terms of speed and torque? **(PreLab Q6)**
7. How much does a single s3003 cost approximately? **(PreLab Q7)**
8. Take the setup of Figure 4.3. Assume Gear A has 60 teeth, Gear B has 18 teeth, Gear C has 48 teeth, Gear A rotates at 5 RPM and Gear D rotates at 40 RPM. How many teeth does Gear D have? **(PreLab Q8)**

9. Read the documentation on the Arduino servo library **servo.h**. You can find it by following this link: `https://www.arduino.cc/en/reference/servo`. In your own words, explain what the 6 functions of the library do and what their in- and outputs are. Write at most 4 sentences per function. **(PreLab Q9)**

10. Solve PostLab Q1. You can test your code by using the already assembled exercise on TinkerCAD: `https://www.tinkercad.com/things/fnHdZKLGPtf`
Make an Autodesk account if you don't already have one, click on "Tinker This". On the top right you will see several buttons. To edit the code on the Arduino press on "Code". To start the simulation press on "Start Simulation" (Figure 4.6). If you correctly solve the exercise you will see the change in width of the PWM pulse on the Oscilloscope. Feel free to play around some more with TinkerCAD and test other electronic components. **(PreLab Q10)**

Note: The Arduino that you can use on TinkerCAD is an Arduino UNO. There are several differences to the Arduino DUE we previously used in the lab: Its Analog Input Pins accept voltages from 0 to 5V instead of 0 to 3.3V. Keep that in mind when you work with the circuit. The analog read resolution of the Arduino UNO is 10 bit and can not be set to 12 bit, as with the Arduino DUE. Additionally the Servo Motor needs 5V to operate, contrarily to the one we would have used in the lab.

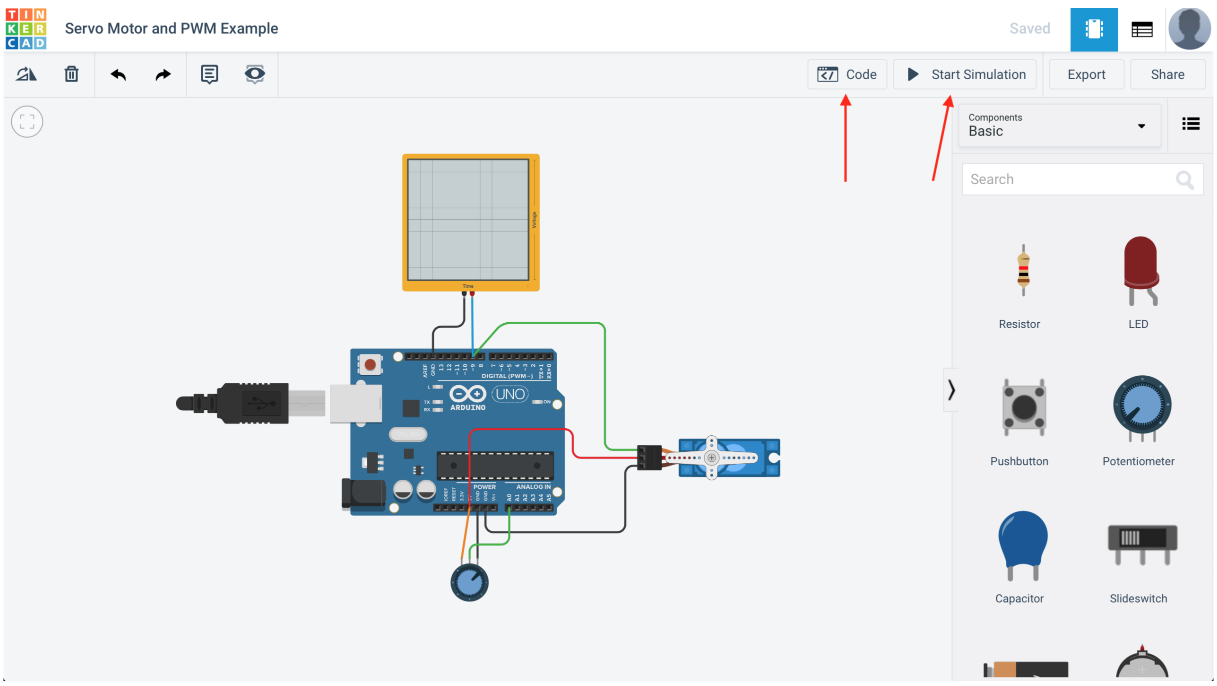

Figure 4.6: TinkerCAD interface. Arrows show the location of the "Code" and "Start Simulation" buttons.

11. Read through the entire lab procedure. The key idea of reading a data sheet is to get a feeling for pinpointing the information of interest and filtering out the rest. If you don't understand something, prepare questions for the lab assistants.

4.3 Lab procedure

Note: Questions marked with **Postlab Qx** have to be answered as part of **PostLab 03**.

Note: Electronic equipment and components can be damaged if input voltage requirements are not met! Please be careful and follow the instructions. If you detect any burning smell, inform the assistant immediately!

4.3.1 Equipment Handling

- Whenever you plug wires in or out make sure to only hold them on the hard plastic part close to the tip. If you directly pull on the wire you risk ripping it out or damaging it in some other way. Take especial care if you pull the Servo out of its socket on the motor shield. make sure to grip both the shield and the black hard plastic end of the cables when you separate them.
- When working with the supplied plastic gears do not worry if there is a loud grinding sound. This is to be expected, as the setup does not have very fine tolerances.
- In order to prevent wires from being ripped out unintentionally keep the breadboard and the servo-box attached to the plexiglas container with the Udoo inside.
- Remember that the analog input pins of the Arduino can manage a maximal input voltage of 3.3 V. If you feed more than 3.3 V to the input you will damage the Arduino board.
- If you have any questions do not hesitate to ask an assistant.

4.3.2 Servo position control via potentiometer

The Adafruit motor shield that is provided in this lab can manage up to two servo motors at the same time. The 3 pins for power, ground and PWM signal are located on the corner over the power inlet of the Udoo and are labeled with "Servo 1" and "Servo 2". The power and ground pins are connected to the power supply and to the ground of the Arduino respectively. The PWM pin is connected to GPIO 10 for "Servo 1" and to GPIO 9 for "Servo 2" (See Figure 4.4 for reference). In this part of the lab you will learn to control the position of the servo using a potentiometer and some code on Arduino.

1. Create a new Lab04 folder, extract the contents of Lab04.zip and put the two mC and mP folders in the new folder. The files contain the skeleton code that you will complete during this lab.
2. Assemble the the circuit in figure 4.7. On your tables you will find a bunch of cables and a rotary potentiometer, which you will need for this part of the lab. You can set the resistance of the potentiometer anywhere between 0 and the maximum resistance by turning its knob. Show your setup to one of the assistants before you connect it to the 3.3 V on the UDOO. Make absolutely sure that you plug it into the 3.3 V and not into the 5 V pin.
3. Plug the servo motors wires into one of the two servo slots of the motor shield.

   Hint: The PWM signal wire (white) goes on the pin that is the furthest away from the power inlet of the Udoo.
4. Open the *Servo_Poti.ino* file. Read the code to understand what the individual servo-related commands and functions are used for.
5. Using your knowledge from previous labs expand the code so that the μC reads the voltage over the potentiometer coming through PIN 0 and maps it to positions for the servo (0V triggers the 0° position, 3.3V triggers the 180° position). Useful functions are:
   $analogRead(PinNumber)$
   $map(x, fromLow, fromHigh, toLow, toHigh)$
   (**PostLab Q1**).

   Hint: By default the resolution for the analogRead() function is set at 10 bit (1024 values), even though it can go up to a resolution of 12 bit (4096 values) on the Arduino Duo. In order to change the resolution add the analogReadResolution(12) command in the setup section. Remember the amount of values that you have when you map them to the 180° for the servo.
6. Now you should be able to set the position of the servo by turning the potentiometer on the breadboard. Notice that the servo is vibrating constantly? Explain what the cause for this vibration is and where it is originating. (**PostLab Q2**)

   Hint: Try visualising the analog input with and without a plugged in potentiometer on the serial monitor. In order avoid a hug of death of the μC by the sheer amount of outputs add a reasonable delay.
7. Describe a method that would reduce the vibrations of the servo while still being able to set its position via the potentiometer. (**PostLab Q3**)

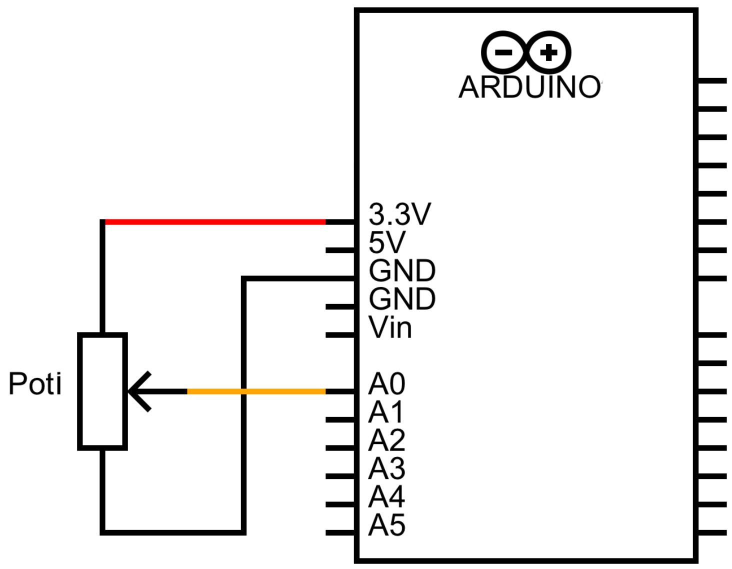

Figure 4.7: Potentiometer and Arduino circuit to set the position of a servo motor.

Bonus 1: Implement your solution to eliminate the jittering. The goal is to have at most one degree of erroneous movement. Make sure that your solution does not have a delay greater than 0.5 seconds between the moment you turn the potentiometer and the servo reacts to it. Explain your solution to an assistant before you try implementing it, so that you do not waste too much time on dead ends. Show your working solution to an assistant. Do not forget to submit your code for the bonus question separately, also label it in a way that makes it clear, that it is the bonus question code.

4.3.3 Torque analysis

1. Consider the situation described in figure 4.8. A blue string is attached to a black disc with radius r and - passing over a return pully - to a green spring with spring constant k. The black disc is fastened to a servo motor with torque T. Using the values: $k = 500N/m$, $r = 0.02m$ as well as the torque of the servo that you can find on the datasheet and neglecting friction, how far should you be able to stretch the spring, before the servo stalls? (**PostLab Q4**)

   Hint: If an external power supply is feeding the motor shield, the same voltage is also supplied to the servo. The motor shield will automatically take its power from the 5V pin on of Arduino if there is no power supply.

2. Build the same experiment as the one displayed on the sketch using the supplied materials. Securely attach the spring to one of the hooks and screw the spindle on the large central gear. Use the circuit which you built in the previous part to deflect the spring as far as possible. Measure and write down the deflection and check if your calculations were correct. There will be deviations from the theoretical value. Calculate the efficiency of the system and name 1 cause for the losses/deviations. (**PostLab Q5**)

3. Use a combination of gears to increase the force that you can apply on the spring, in order to get at least 1.5 times the deflection that you had before. The amount of teeth are engraved on the gears, so you do not have to count them. Write down the calculations to determine the adequate gear multiplications and implement it on the servo box. Keep in mind that as you increase the torque you also reduce the maximum run length of the system. Show your results to an assistant. Do not worry if you get slightly different values than your neighbouring teams, as the setups can and will vary. (**PostLab Q6**)

   Hint: Remove the spindle from the big 60 teeth gear and attach it to the 48-teeth gear with the 4 threaded holes in it. Do not apply too much force when attaching the spindle to the gear, as you may break the threads in the plexiglas gear. When you have determined an adequate gear ratio use the red pin with the screw head to firmly attach it to the white box. The screw head will prevent the gear from being lifted off the pin. Use the small spacer discs to position the gears at the correct height.

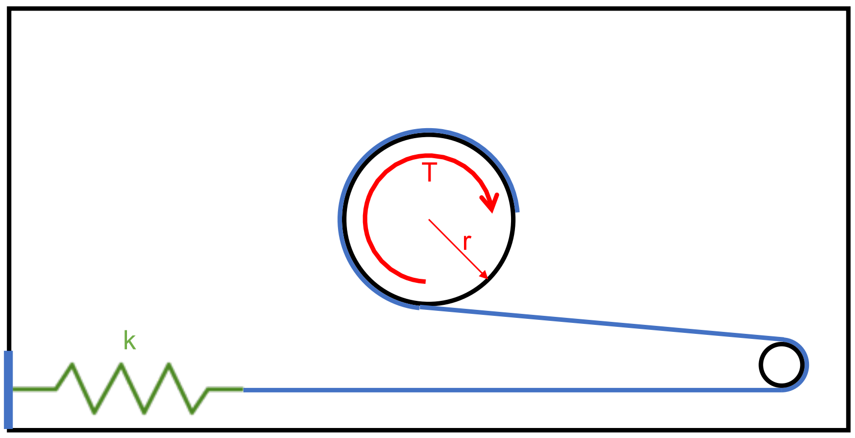

Figure 4.8: Torque analysis schematic.

4.3.4 Servo Timing Watch

The last task of this lab is to build an analogue timing watch, with a hand to display the seconds (0-60s) and a second hand to display the minutes (0-2.5min). Both will be driven simultaneously by the servo located in the center. A schematic can be seen in Figure 4.9. As the range of motion of the servo is limited to 180° your task is to find a combination of gears so that the timer can run for a full 2.5 minutes when the 180° are used up. Additionally you will complete the skeleton code in order to control the servo using the serial port. The idea is that you can set the amount of seconds you want your timer to run on a terminal window and then, every second a signal is sent to the Arduino to advance the watch hands by one second.

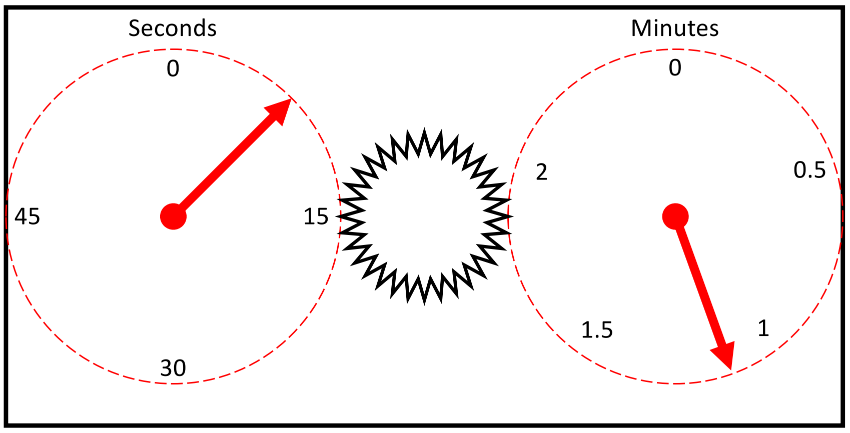

Figure 4.9: Schematic of the timing watch.

1. Calculate the gear ratios for both hands that are necessary for the watch to run for 2.5 minutes by rotating the servo motor 180°. (**PostLab Q7**)

   Hint: 1 minute represents a full rotation of the seconds watch-hand and 2.5 minutes represent a full rotation of the minutes watch-hand.

2. Calculate the theoretical resulting torque at both of the watch-hands. (**PostLab Q8**)

3. Build the clock using combinations of the gears that are provided. Write down the combinations that you have used. (**PostLab Q9**)

   Hint: The watch hands are the red, pointy pieces with two pins and a hole in the middle. Pin down the watch dials under the spacer discs to prevent them from moving around when the gears move. Align the watch hands with the "0" position of the dials.

4. Open the *Servo_Timer.ino* file in the μC folder as well as the *timer.c* file in the μP folder.

5. Look at the function $delay()$ at the top of the *timer.c* file. It is used to precisely time 1 second before sending the command to move to the Arduino board. Explain how it works. (**PostLab Q10**)

   Hint: The function $clock()$ from the library *time.h* returns the amount of microseconds elapsed since the program was started.

6. Expand the skeleton code in the *timer.c* file so that when the program is executed it has the following functionalities: (**PostLab Q11**)

   (a) At startup: ask the user how long you want the watch to run (between 0 and 150 seconds) or if you want to quit the program.

   (b) If a suitable number has been entered send a signal to the Arduino to reset the servo to its starting position. After this send a signal to the Arduino precisely every second through the serial port to tell it to advance the timer by 1 second. Repeat this until the set amount of time is reached.

   (c) When the timer is done return to the startup query, where you decide whether to continue with another timer or to quit the program.

7. Expand the skeleton code in the *Servo_Timer.ino* file and upload it to the Arduino so that it has the following behaviour: (**PostLab Q12**)

   (a) Calculate the angle that the servo has to move in order for the watch hands to advance by 1 second each time an input is received. Note that the previously used $servo.write(angle)$ command only works with $integers$. In order to have a higher resolution use the command $servo.writeMicroseconds(lengthOfPulse)$. This command directly sets the length of the PWM for the servo, thus allowing for a higher precision. Use the max and min PWM-values from the datasheet for the $0°$ and $180°$ positions.

   (b) In the setup section, initialize the declared variables, and position the servo at its starting position ($0°$).

   (c) The 2 different character inputs "a" and "b" that you send from the C-code are read through the serial port: One input resets the servo to its starting position, the other input triggers the servo motor to advance one step corresponding to one second. Implement these two behaviours in the loop section.

   Hint: Be careful not to add up rounding errors, as it also only works with $integer$ values and the angle that you calculated will have decimal points and should be in the $float$ format.
   Example: Assume the servo step for 1 second is $2.6°$. As an int this will be represented as $2°$. If you add the next step to the rounded $2°$ you will get $4.6°$, which again, as an int will be rounded to $4°$. The actual angle however would be $2*2.6°$, or $5.2°$, resulting in the more accurate rounded integer value of $5°$.

8. Show your working system to an assistant who will time at your timer clock.

4.4 Postlab and lab report

1. Show your working system to the teaching assistant.

2. Upload a single PDF-file with your solution to moodle. The file should contain your answers to the postlab questions Q0-Q8 as well as the code in the $\mu C.ino$ and $\mu P.c$ files. Please upload your solution in time (before next lab session), late submissions will not be corrected.

3. Print the PDF-file and hand it in at the beginning of the next lab session.

4. Come prepared with the Prelab procedure for the next lab.