Characterization of a $^{85}\text{Rb}$ magneto-optical trap

Mathijs de Bruin (0420891)

Supervisors:
Ben van Linden van den Heuvel
Richard Newell

August 2, 2009
Thanks

During the course of this project I have had a great deal of help from a lot of people in and around the WZI institute, without whom my bachelor’s project would have been guaranteed to end up a disaster. This is why I would hereby like to thank all of the people that gave me help, guidance and advice along the way. Specifically I would like to thank Ben van Linden van den Heuvel for giving me the chance to do this project in the first place and Richard Newell for his many efforts in helping me and for his sick humor which kept me alive during many days of failed laser stabilization. Furthermore, I would like to thank the following people individually for their help and good advice, which was there when I needed it: Carolijn van Ditzhuijzen, Antje Ludewig, Shannon Whitlock and Philipp Wicke.
Abstract

We present the study and measurement of the typical parameters of a Rb\textsuperscript{85} magneto-optical trap using fluorescence imaging. We report an average temperature of 83 ± 6 µK and an average FWHM cloud size of 705 ± 118 µm. Also, we found an average decrease in temperature of 2.93 µK/hour.
1 Introduction

In the following report the basic principles and technology underlying atomic cooling and trapping using a magneto-optical trap (MOT) are described. A MOT is essentially a combination of specifically tuned laser light combined with a quadrupole magnetic field. This allows for the cooling and trapping of atoms from (typically) a hot source. Using this technique atomic velocities corresponding to temperatures as low as 1 \( \mu \text{K} \) can be routinely achieved. [4; 10; 11]

Specifically, we studied the typical parameters of an \( ^{85}\text{Rb} \) MOT setup and measured the temperature and FWHM of the cloud (along the horizontal axis). For these measurements, special software has been developed to perform real-time image processing and analysis. Also, a combination of software and hardware was set up as to allow for accurate timing and control of the experiment.

In the experimental setup we performed our measurements on, research was done in long-distance dipole-dipole interactions between ultracold Rydberg atoms. In order to be sure that what was measured were indeed dipole-dipole interactions, we needed to know whether the velocity distribution of the atoms did not allow them to interact elastically. This is why a temperature measurement was necessary.

The basic structure of this report is in three parts. It starts out with a theory section (section 2) for readers unfamiliar with the basic working principles of a MOT and the characterization thereof. In the second part (section 3) we present the specifics of the actual measurement performed. The third part (section 4) lists and briefly discusses the outcome of the experiment.

Besides the main content of the report there are two appendices listing the source code of the software written for this setup to perform real-time image processing and analysis (Appendix A) and timing and control of the actual experiment (Appendix B). This software is licensed under the GNU General Public License and consequently, the full GPL is distributed along with it in Appendix C.
2 Theory

2.1 Laser cooling and trapping

In 1975 laser cooling was simultaneously proposed by two groups of researchers, the first one led by Wineland and Dehmelt, the second by Hänsch and Schawlow. This principle was first demonstrated in 1978 by Wineland, Drullinger and Walls for atoms along one dimension. It was extended in the 80’s by Chu and his coworkers, who developed a technique to cool a cloud of atoms along all three dimensions commonly referred to as ‘optical molasses.’ For developing and perfecting this technique Chu, Cohen-Tannoudji and Philips were awarded the Nobel Prize for Physics in 1997.[2; 3]

Optical molasses was then further extended to the MOT or magneto-optical trap, which allowed for cooling and trapping of even colder atoms using six circularly polarized laser beams and a magnetic field as in Figure 1. These techniques will be further explained in the following sections.

The availability of such cooling techniques allowed for experimental research into ultracold atoms and quantum gases, such as Bose-Einstein condensates. This in turn brought up optical latices, matter wave interactions and in the near future quantum computing as experimental research topics.

![Figure 1: Schematic display of a typical MOT setup, showing the cooling beams, the repumper beam and the anti-Helmholtz magnetic configuration. In this figure the curved arrows following the shape of the laser beam signify left or right handed circular polarization, the black lines through the coils indicate the structure and direction of the magnetic field and the arrow along the coil denote the coil current $I$. Taken with permission from [7].](image)

2.1.1 Doppler cooling

Laser cooling uses the fact that whenever an atom becomes excited by absorbing a photon, its momentum will be transferred to the atom. When that atom falls back into a lower energetic state, it will do so by emitting a light quantum in...
a random direction. Now if we repeat this process numerous times\(^1\) by using a beam of photons, we will observe the atom to gain net momentum in the beam direction. This happens because an atom will emit photons (and thus momentum) in random directions but absorb them from only one.

If we take into account that an atom only absorbs photons with energies corresponding to transitions in the atom’s energy spectrum, we can use the Doppler shift\(^2\) to tune the wavelength of laser light such that only a certain velocity group interacts with it. In this way we can slow down atoms by impending red shifted laser light in the opposite direction of their motion. Similarly, detuned laser beams from opposite directions can be used to slow down atoms with regards to a single axis.

However, if we actually want to cool atoms we will have to slow them down in all directions. This requires an experiment with opposing laser beams on all three axes, leaving us six beams in total as displayed in Figure 7. Using this setup we can effectively cool down gaseous clouds of atoms in vacuum to very low temperatures. This technique is optical molasses, in reference to the molasses-like effect the atoms experience in the intersection of the laser beams.

![Figure 2](image)

**Figure 2:** Schematic display of the beam path in a typical MOT setup. The detuned laser beam is splitted and focussed such that the six cooling beams and the one repumper beam cross to form a MOT. Taken and modified with permission from [7].

### 2.1.2 Repumping

We have thus described a setup able to cool atoms using carefully tuned laser light but this only works for atoms in the ‘cooling state’, the specific hyperfine state the laser is tuned into. Because a small fraction (roughly 0.1\%) of the atoms fall back\(^3\) to a different state, we require some mechanism to guarantee that our cooling state remains sufficiently populated.

To achieve this we typically use atoms whose hyperfine ground states have a degeneracy of two, such as alkali atoms. Assuming one ground state pertains to our cooling transition, we can tune a second ‘repumper’ laser to the transition from the other ground state to a higher excited state which likely decays into the cooling state. These transitions are displayed in Figure 3 for \(^{85}\text{Rb}\), which is

---

1. Allowing for multiple Rabi-cycles to occur.
2. Where moving atoms perceive the photons shifted in frequency.
3. Atoms generally decay at the Rabi frequency, on the order of 1 Mhz.
used in our experiment. Now whenever an atom decays into the other ground state, it will immediately be excited and either decay back into the cooling state or into the other state and become excited once again.

This process is called repumping and insures that the ground state pertaining to our cooling transition remains sufficiently populated throughout the cooling process.

Figure 3: Schematic display of the relevant MOT transitions in $^{85}\text{Rb}$. The nuclear spin $I = 5/2$ causes the hyperfine splitting depicted by $F$, with the hyperfine shift $\Delta \nu$ in MHz. Taken with permission from [7].

2.1.3 Trapping

There remains one problem however; when we have slowed down an ensemble of atoms such that it does not interact with the detuned laser light anymore, the atoms will still exhibit random motion due to their little remaining velocity. This random walk of individual atoms will cause an average square displacement as a consequence of which we will gradually lose the coolest atoms from our ‘trap’.

This is where magnetic trapping comes into play. By putting the cooling setup in a so called anti-Helmholtz magnetostatic configuration, which is zero in the center of the trap and increases towards the edges, we attain a position dependent Zeeman splitting of the atom’s energy spectrum. Effectively, this widens the cooling transition as a function of the distance from the center of the trap. If we now use a more intricate setup where all six laser beams have a specific circular polarization, we obtain an effective trapping force because interactions of displaced cool atoms occur only with the laser beam coming from the direction of their displacement. This now constitutes a MOT or magneto-optical trap.
2.1.4 Doppler cooling limit

Using the previously discussed method we can cool clouds of atoms to extremely low temperatures. Nevertheless, there is a theoretical limit to the temperature we can cool to by transfer of momentum from photons.

After an atom has absorbed a photon from the cooling beam, it decays back to its ground state by emitting one in a random direction. The emitted photon however, carries momentum and thus an equal magnitude of momentum carries over to the atom. Although the momentum increase to individual atoms is low, the heat thus generated poses a fundamental limit to Doppler cooling.

The lowest temperature that can thus be reached is the Doppler limit:

\[ T_D = \frac{\hbar \gamma}{2k_B} \] (1)

where \( \gamma \) is the natural linewidth of the cooling transition, \( m \) is the mass of the atom and \( k_B \) is Boltzmann’s constant. Typical values for the Doppler limit are of the order of 100 \( \mu \)K, which is roughly what we measured in our experiment.

2.1.5 Sub-Doppler cooling

Temperatures below the Doppler limit can however be reached, by using sisyphus or evaporative cooling and other techniques.

Using these techniques, temperatures as low as the recoil limit can be reached. This is the temperature that remains due to the momentum caused by the ‘spontaneous’ emission of photons by atoms in an electromagnetic field. The recoil temperature is given by

\[ T_r = \frac{\hbar^2 k^2}{2mk_B^2} \] (2)

where \( k = 2\pi/\lambda \), the magnitude of the wave vector, \( m \) is the mass of the atom and \( k_B \) is Boltzmann’s constant. Typical values for the recoil limit are of the order of 1 \( \mu \)K.

2.2 Diode lasers, frequency tuning and stabilization

The laser cooling techniques described in the previous section require precise and stable tuning of the wavelength of the laser light to coincide with our required ground state transitions. For our experiment we have relied on widely available near-infrared Rubidium diode lasers. As the free running frequency stability of these lasers (in the order of 100Mhz) is generally inadequate for our purposes, an active stabilization method is required.

Generally, the intensity and frequency of diode lasers depend on the temperature, the configuration of an optional reflection grating and the current driven through the diode. Usually the temperature is controlled by a thermostat working on a heatsink connected to the diode casing, the grating is controlled by a combination of a screw and a piezo-electric actuator and the current is controlled by the laser’s power supply.
In our experiment we used a grating in the Littrow configuration, where the laser cavity is extended by a reflective grating which reflects the first order interference back to the rear facet of the diode to generate a much narrower linewidth (in the order of 1MHz). The angle of the grating allows us to tune the frequency of the laser and can be coarsely rotated by a screw and finely controlled by a piezo actuator. This allows us to scan over a range of frequencies and perform spectroscopy in order to tune the lasers to desired transitions.

2.2.1 Doppler-free saturation spectroscopy

A form of absorption spectroscopy is employed in order to obtain a signature of the interaction of the laser output with the particular atoms under study and thus to generate the desired locking signal. However, the spread in random motion of the gaseous atoms in the cell causes a Doppler broadening of the spectral lines; the moving atoms perceive and resonate at different frequencies with the laser light. As a consequence of this it becomes impossible to discriminate the fine and hyperfine spectra of the atoms, severely limiting our ability to tune the laser to the desired transition. An example of a Doppler-broadened signal for Rubidium can be seen in Figure 5a.

Figure 4: Schematic display of a Doppler-free saturation spectroscopy setup used for stabilizing the cooling laser’s frequency in the upper rectangle, where a probe and a pump beam are sent through a Rubidium cell in opposite directions. The probe beam leaves the cell to hit a detector, while the pump beam ensures saturation of the moving atoms causing Doppler broadening. Taken and modified with permission from [7].

To overcome this problem we can use a technique called Doppler-free saturation spectroscopy, a setup for which is schematically displayed in Figure 4, where we use an overlapping ‘pump’ beam in the opposite direction to saturate the excited atoms that are stationary along the axis of beam propagation. This works because the moving atoms in the overlapping area resonate with either one of the beams,\textsuperscript{4} while the atoms at rest resonate with both. The result of this will be a saturation of the ground state of the atoms due to the pump beam,\textsuperscript{5} and thus a peak in the transmission at the exact frequency of transitions because there is little or no absorption there, as is shown in Figure 5b. Around

\textsuperscript{4}They can either move towards a beam or away from it, but not both.

\textsuperscript{5}The pump beam will remain invisible for moving atoms in as far as the spectroscopy is considered.
the transmission peak however, we will still see the Doppler broadened peaks unchanged. \[1; 5\]

2.2.2 FM spectroscopy

Using Doppler-free spectroscopy in combination with the piezo actuator, we can perform scanning spectroscopy with sufficient detail. But this alone will not keep the laser locked to a specific transition. To achieve this we use a different setup to detect minute deviations in the laser frequency, called a Pound-Drever-Hall detector.

In such a setup we modulate the driving current for the laser with an RF signal on the order of 10 MHz. This amplitude modulation of the current causes a frequency modulation in the laser light, as shown in Figure 6. In turn, this equals a ‘scan’ over the atomic spectrum when looking at the Doppler-free spectroscopy signal.

Because the amplitude of the transmission signal equals the difference in transmission between the lowest and the highest frequency turning points, it approaches the derivative of the absorption spectrum. The amplitude is simply obtained by demodulating this AM signal.

To lock the laser to a certain dip in the absorption spectrum, we simply modulate the current so the frequency ‘scans’ near the dip. We can use the demodulated transmission signal to provide negative feedback for the offset of the modulated current and thus keep the laser locked steadily to the transition’s frequency. \[6\]

2.3 MOT characterization

Several methods exist for characterizing a MOT by determining its typical parameters. One such methods is absorption imaging, where an extra laser beam is shone through the MOT, the image of which can be captured using a camera.

However, in our setup we were not able to set up an extra laser path through...
the vacuum chamber (due to a lack of space and material). For this reason we chose to use fluorescence imaging, where we monitor the light fluorescing from atoms as they fall back into their ground state. [9, sec. 4.3.1]

This light can also be imaged using an ordinary camera, which can be used to determine the following parameters with varying precision:

- Number of trapped atoms
- Loading and loss rates
- Relative cloud dimensions (FWHM)
- Temperature

Of these parameters we have attempted to estimate the trapping and decay rates and the number of atoms. Furthermore, we performed precise measurements on the temperature through the FWHM of the cloud. The determination of these parameters shall briefly be discussed below.

2.3.1 Number of trapped atoms

The total number of atoms in a MOT can be calculated from $F$, the total amount of fluorescence photons emitted by it divided by $R$, the fluorescence per atom:

$$N = \frac{F}{R}$$

(3)
We can calculate $F$ if we take into consideration that the atom cloud is not optically dense and that it scatters light isotropically. This way we know that the fraction of light that falls onto a detector with lens radius $r$ at a distance $d$ from the MOT is the surface fraction with radius $r$ of a sphere with radius $d$.

Now taking $P$ as the total laser power, we can calculate the number of photons

$$F = \frac{hc}{\lambda} \frac{4d^2}{r^2} P$$

where $h$ is Planck’s constant, $c$ the speed of light and $\lambda$ the wavelength of the fluorescing light.

Continuing, we need to know the amount of photons scattered per atom for a specified transition. This is given by the following equation

$$R = \frac{\Gamma/2}{1 + I/I_s + 4(\Delta/\Gamma)^2}$$

where $\Gamma$ is the natural linewidth of the transition, $I_s$ is its saturation intensity, $\Delta$ is the detuning of the laser from the transition and $I$ is the total intensity of the laser beams which are exciting the atoms.

However, it turns out to be very hard to give a good estimate of $\Gamma$. Often, we have a relative uncertainty in the outcome that approaches the magnitude of the measured value. Therefore most experimenters choose to determine the atom number by absorption imaging and use this to gauge subsequent fluorescence measurements. In our particular experiment we had no absorption beam setup, leaving us with only a very crude estimate of the number of atoms in our cloud.

### 2.3.2 Loading and loss rates

Depending on the strength of the trapping force and the partial pressure of the trapped atoms, there is a certain rate at which a MOT traps or loses atoms. These are the loading and loss rates, which are equal for a MOT in equilibrium (since the amount of atoms we lose is equal to the amount we are gaining).

To determine this property we measure the total amount of atoms as a function of time while we turn on the trapping force. This yields a specific growth curve for which the maximum of the derivative represents the loading and loss rates of a stable MOT.

As the error in the atom number propagates to the loading and loss rates as well, this time again we had to suffice with a crude estimation.

### 2.3.3 Cloud temperature

Measurement of the temperature of extremely cold atoms is very difficult, since even the slightest perturbation of the system will generate heat. Luckily, using fluorescence imaging there is a fairly easy way to measure the temperature in a MOT; we can measure the free (or ballistic) expansion of the atom cloud due to random thermal motion of the atoms.

Using this method, often referred to as ‘release and recapture,’ we switch of the cooling and trapping forces during some known amount of time, after
which we switch on the laser to image the fluorescence signal which we use to determine the cloud size. Repeating this process while scanning over the amount of time we allow for free expansion we can perform precise (average) temperature measurements for the atom cloud in a MOT setup.

To measure the cloud size, we assume the spatial structure of the cloud to fit a Gaussian profile,[9, sec. 2.3] so the density function fits

\[
n(r, z) = n_0 \exp\left(-\frac{r^2}{2\sigma_r^2} - \frac{z^2}{2\sigma_z^2}\right)
\]  

where \(n_0\) is the peak atomic density, \(r = \sqrt{x^2 + y^2}\) and the cloud widths \(\sigma_r^2, \sigma_z^2\).

Furthermore, we know that the temperature of a system of atoms is directly related to the average velocity of the atoms in that system by

\[
\bar{v}^2 = \frac{3k_B T}{m}
\]

with \(m\) the atomic mass, \(k_B\) Boltzmann’s constant and \(T\) the temperature.

From this it follows that the cloud size \(\sigma(t)^2_{r,z}\) grows linear over time and thus

\[
\sigma(t)^2_{r,z} = \sigma(0)^2_{r,z} + \frac{k_B T}{m} t^2.
\]

which can be easily measured by capturing the fluorescence profile over one axis.
3 Experimental setup

In our experiment we have measured the temperature of the MOT according to the method described in section 2.3.3. The current chapter will detail on the specifics of the actual measurements and the setup they were performed upon.

![Figure 7](image)

**Figure 7:** An exploded view of the vacuum chamber setup used to create a MOT with the cooling beams in red, the repumper beam in yellow and the magnets in brown. Taken with permission from [7].

3.1 MOT setup

We have measured the temperature of a cloud of atoms in a magneto-optic trap as described in section 2.1. An exploded view of the specific setup we performed our measurements on is shown in Figure 7. In this setup we used a Toptica DL 100 at 780 nm as cooling laser and a Toptica TA 100 at 795 nm as the repumper laser.

We performed locking using two Pound-Drever-Hall detector setups, as described in section 2.2, through which we diverted a small fraction of the laser light. For spectroscopy we used two heated $^{85}$Rb tubes.

The cooling and repumper lasers were locked respectively to the $5S_{1/2} (F = 3) \rightarrow 5P_{3/2} (F = 4)$ and the $5S_{1/2} (F = 2) \rightarrow 5P_{1/2} (F = 3)$ transitions of $^{85}$Rb. This is also shown in Figure 3 on page 5. The cooling laser was then detuned 12 Mhz using an acousto-optic modulator (AOM) as to allow for Doppler-cooling of the Rubidium atoms.

3.1.1 Hardware setup

In order to perform the temperature measurement, we discovered that we could switch the AOM used for detuning the cooling laser on and off by switching DC voltages on its modulation ‘video’ input. Also, the anti-Helmholtz coils were connected to a switching unit allowing us to switch them on and off using a TTL signal.
To measure the cloud size after allowing for expansion we used a DCAM compatible IEEE-1394 (Firewire) infrared-camera for which we wrote extensive monitor, measure and capture software using MATLAB.

The camera used, a Prosilica EC750, has an external trigger input with which it is possible to trigger a frame capture using a TTL signal. The used software and its functioning will be further discussed in section 3.3.

To perform our measurements with precise timing we used an SRS DG535 Digital Delay Generator, which allows for precise control of timing (5 ps resolution) by generating pulses and delays on 4 channels. Because of the technical limitations of this device, we also used a function generator to generate a longer block pulse upon triggering by the delay generator in order to fully switch off the magnets during the release and capture measurement. More explanation on the timing and control of the experiment will be given in section 3.2.

### 3.1.2 Growth curve and loading rate

To assertain enough time is available for a MOT to become fully loaded on each capture and recapture cycle, it was necessary to measure the growth curve of the MOT. This growth curve is displayed in Figure 3.1.2 and shows a total loading time of about 7 seconds.

![Figure 8: Growth curve of the MOT. The atom count is normalized by assuming that the total atom number in the trap is equal to the estimate of $1.8(4) \cdot 10^7$ measured on this setup in [7]. The equilibrium loading and loss rate thus found roughly equals $3 \cdot 10^6$ s$^{-1}$.](image)

In order to perform this measurement we used the method described in section 2.3.2, which allowed us to roughly determine the equilibrium loading rate as well. Using the atom count specified in [7] we are able to conclude that
the loading rate is roughly equal to $3 \cdot 10^6$ atoms per second. However this is merely an estimate as we do not know whether the atom count is indeed the same and even in this case, there is a large error in the atom count propagating to the loading rate.

### 3.2 Timing and control

To perform accurate temperature measurements using the ‘capture and recapture’ method described in section 2.3.3, it is essential to have accurate control over the timing of the experiment. We accomplished this by remotely programming a DG5353 Digital Delay Generator using a Python program over GPIB in concordance with a digital function generator to overcome a functional shortcoming in the DG535.

With use of the delay generator we can designate a total of four moments in time (henceforward called A, B, C and D) relative to a reference time ($T_0$), as shown in the timing graph in Figure 9. On this machine there are outputs which receive a short pulse corresponding to each of the specified moments (including the reference $T_0$). Besides this there are outputs which give a pulse that starts at A and ends at B (AB) and one with the same functionality for C and D (CD). For AB and CD there are also inverted outputs.

**Figure 9:** Schematic display of the timing used for the temperature measurement. Here $t_{\text{delay}} = 31$ ms is the trigger delay for the CCD camera, $t_{\text{scan}}$ is the variable exposure delay over which we scan in 0.05 ms steps, $t_{\text{expose}} = 2$ ms is the exposure time during which the cooling laser is temporarily switched on again and $t_{\text{measure}} = 11$ ms determines the range over which $t_{\text{scan}}$ runs as well as the shutter time of the camera (which is set to $t_{\text{measure}} - 1$ ms as an extra margin). Furthermore, $t_{\text{wait}} = 8$ s during which we wait for a new cloud to form to repeat the measurement for the next step in the scan. For the exact parameters used in the experiment, please refer to Listing 2 in Appendix B.
Figure 10: Schematic display of the triggering setup used. We used a Python program to control a DG535 Delay Generator which directly switched the AOM to attenuate the cooling laser power (by \( nn \) dB). Furthermore, the generator triggers an image capture by the CCD camera and a function generator switching off the magnets to allow for ballistic expansion. For the exact parameters used in the experiment, please refer to Listing 2 in Appendix B.

Each delay and pulse output has an independently adjustable offset and amplitude which can be set between -3 V and 4 V with 10 mV resolution. The maximum transition for each output is limited to 4 V. In addition, one can also separately select 50 Ω or high impedance termination for each output. Preset levels, corresponding to TTL, NIM and ECL, can also be selected.

As we wanted to be able to do very fast switching, we have connected the DG5353’s outputs directly to the AOM ‘video input’. We found that switching a pulse with an offset of -0.31 V and an amplitude of 0.36 V gave a measured damping of roughly 18dB.

Because the delay generator only allows for two time spans to be set, we required a different way of switching the CCD camera and the magnets according to the timing scheme shown in Figure 9. For this we used a digital function generator set to give out a block pulse that switches off the magnets while the release and recapture cycle runs. The CCD camera could be set to have an exposure time the length of the full cycle and is triggered on \( T_0 \) as to account for the camera’s shutter delay, which was measured to be roughly 31 microseconds as shown in Figure 11.\(^6\) The wiring scheme of this timing and control setup is shown in Figure 10.

To be able to control the experiment we wrote an almost complete Python wrapper around the delay generator’s GPIB remote control interface. Using this wrapper it, which is fully listed in appendix B, we could manually trigger measurement cycles (as to overcome the DG535’s limitation of a total maximum of one second) and implement the scanning functionality required for the experiment.

\(^6\)This delay was measured using a LED in front of the lens, connected to the AB output of the delay generator. The camera trigger was connected to the \( T_0 \) output. By measuring the total signal from the camera in a darkened room while scanning over the time span between \( T_0 \) and \( A \) we could clearly discriminate the camera’s shutter delay.
Figure 11: Measurement of camera trigger delay for a shutter time of 5 ms. A trigger was given at $t = 0$ ms. When the average pixel value is high, the shutter is open.

3.3 Monitoring, analyzing and capturing images

Measuring the width of an atom cloud using the image from a CCD camera required an extensive amount of image processing and analysis; because of the amount of data otherwise generated and the minimal amount of extra effort required, we chose to do most of this processing in real time. Because there was little to none software available to do this kind of processing, a custom piece of software and a suitable GUI (graphical user interface) were developed as to allow for a dynamic range of applications and uses. The source code of this software is licensed under the GNU General Public License (GPL) and is completely contained in appendix A, where the GUI and functional structure of the program are also described. The GPL is distributed along with the software as is required by this license. It is contained in appendix C.

The software allows for adjusting the exposure time in between capture sessions and enables switching between continuous capturing (where the framerate in limited only by processing speed and the exposure time) and capturing on an external triggering signal (where the camera waits for a trigger pulse to capture each single frame). Besides these features the software had to be able to perform the following tasks in real time in order to perform the temperature measurement:

- Capture images in continuous or external triggering mode.
- Display the raw camera image using a visual contrast-enhancing colour scaling.
- Use a previously captured image to do background subtraction in order to increase the useful data in the capture signal.
- Allow for adjustment of the dynamic range, cropping and zooming of the
- Perform fitting of a Gaussian curve to the summed signal for one axis of the data in order to measure the cloud width and the (relative) amount of atoms.

- Calculate a $\chi^2$ error value for the fitted values.

- Measure and display the minimal, maximal and average signal in the current image in order to prevent distortion in the image.

- Display historical data over the course of the running experiment in a graph as to watch the development over time.

- Write the measured parameters to a preselected file on a disk.

- Write the captured images in sequence to a specified folder on a disk.

The role this software played in the entirety of our measurement process is shown in Figure 12a, from which it should also become clear what parts of the processing could not be performed in real time; namely those which concern and interrelate the results of full scans over the expansion of atom clouds. This part of the measurement will be described in the next section.

**Figure 12:** Schematic display of realtime- and post-processing flows used for the measurement.
3.4 Data processing and analysis

After a full scan has been made, the resulting data was processed using the steps shown in Figure 12b using the Mac OS X program Plot.[8] We will now describe the steps performed when processing this data.

In order to make fits of the data we first made cuts so as to limit the influence of unknown parameters such as Eddie currents in the metal structure upon switching off the magnetic field. These cuts were made manually made on a best-effort basis and seem to be very effective in enhancing the quality of our fits.

After these cuts were made, the measured data was scaled according to the proportions given in [7]. The error therein was taken into account when we fitted the data to the formula discussed in section 2.3.3, written as

\[
\sigma(t) = \sqrt{\sigma(0)^2 + \frac{k_B T}{m} t^2}
\]

where \(k_B\) and \(m\) were specified and \(T\) as well as \(\sigma(0)\) are fitting parameters.

Because the overall fitting error turned out to be negligible with regards to the statistical variance between individual measurements, we decided to only discuss the statistical error we found in our experiment. Since this formula only depends on the growth of the cloud size, we do not expect significant sources of systematic errors.
4 Results and discussion

4.1 Results

The results obtained fall remarkably well within the temperature limit expected by the research group on the basis of the setup design. We performed a total number of 18 temperate measurements corresponding to scans of roughly 26 minutes in length, the results of which are shown in Table 4.1. On average, we measured a temperature of $83 \pm 6 \, \mu K$.

In Figure 4.1 two graphs show the respective first and second measurements performed on the 12th and the 13th of November, 2008. These graphs represent exemplary picks for the temperature measurement scans performed. In these graphs, the raw measurements of the cloud width are shown as light grey lines. The black lines show the fitting error therein, averaged over 50 points. The red line is the best fit, corresponding to $92.84 \, \mu K$ for the first and $84.50 \, \mu K$ for the second measurement.

<table>
<thead>
<tr>
<th>Date</th>
<th>Batch</th>
<th>T ($\mu K$)</th>
<th>FWHM ($\mu$m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-11-2008</td>
<td>1</td>
<td>93</td>
<td>850</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>89</td>
<td>851</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>89</td>
<td>822</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>91</td>
<td>841</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>84</td>
<td>840</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>86</td>
<td>833</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>90</td>
<td>833</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>82</td>
<td>814</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\mu$</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sigma$</td>
<td>3</td>
</tr>
<tr>
<td>13-11-2008</td>
<td>1</td>
<td>83</td>
<td>584</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>84</td>
<td>564</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>83</td>
<td>588</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>82</td>
<td>597</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>85</td>
<td>586</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>73</td>
<td>616</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>77</td>
<td>626</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>77</td>
<td>605</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>70</td>
<td>615</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>72</td>
<td>628</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\mu$</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sigma$</td>
<td>5</td>
</tr>
<tr>
<td>Overall</td>
<td>$\mu$</td>
<td>83</td>
<td>705</td>
</tr>
<tr>
<td></td>
<td>$\sigma$</td>
<td>6</td>
<td>118</td>
</tr>
</tbody>
</table>

*Table 1: Measured temperatures and initial horizontal full width half maximum of the cloud and their respective statistical average $\mu$ and standard deviation $\sigma$. 
(a) Measurement performed on the 12th of November, 2008 around 15:00. It is shows clearly that the expansion of the cloud matches closely with the theoretically expected behaviour.

(b) Measurement performed on the 13th of November, 2008 around 12:00. As the graph shows, the fitted curve deviates a fair amount from the values fitted to it. Also, the (relative) atom count for this measurement was much higher.

**Figure 13:** Examplary picks from two actual measurement scans.
4.1.1 Long-term temperature decrease

Upon comparing the separate measurements we noticed a long-term downwards trend in temperature, as is shown in Figure 4.1.1. In our setup, a typical measurement scan takes approximately 26 minutes over which we measured a decrease of 0.99 µK on the 12th and 1.55 µK on the 13th of November, averaging to a decrease of 2.93 µK/hour (wherein a large error is to be expected).

After comparison with the initial FWHM of the cloud we discovered no apparent linear or other temporal behaviour and found no relationship to the suggested long-term temperature decrease.

![Figure 14: Long-term temperature decrease fits for the two measurement batches on the 12th (upper red line) and 13th (lower blue line) of November, 2008 displaying an average decrease of 2.93 µK/hour.](image)

4.2 Discussion

The research group normally working with the characterized MOT setup assumed the average temperature to be less than 100 µK. With an average temperature of 83 ± 6 µK this falls well within the expected values. Also, the measured expansion yielded a nearly perfect fit and the statistical error in the measurement batches was sufficiently low to assert self-consistency.

However, systematical errors either due to unforeseen flaws in the measurement procedure or to calibration errors might still exist. The first could be prevented by cross-reference with another measurement method, the second by recalibrating the values assumed to be known (ie. setup dimensions, camera pixel scaling).

With regards to the long-term temperature decrease there is no definite
explanation as of yet. However, we suggest that the continuous release and recapture cycles could favour the loss of hotter atoms since they have a higher chance of ‘flying off’ while the trap is off.

One possible way to test this hypothesis is by having the MOT run for longer periods of time before performing a temperature measurement. We could ‘scan’ over the running time of the MOT with only a single measurement cycle each time. If we would still see the temperature decrease we can be sure that this explanation is not sufficient.
5 Conclusion

We have successfully performed a temperature measurement for a Rb$^{85}$ magneto-optical trap using the capture and recapture method. In the process we have made an extensive study of laser cooling, laser stabilization and general MOT theory. Also, specialized software was developed to perform the experiment which should be useful for this and other similar experimental setups in the future.

The results of the temperature measurement itself are self-consistent and accurate and fall well within the existing expectations of the research group that built and designed the setup. However, systematical errors have not been taken into consideration and should be considered for an extension of the current experiment.

We did notice a remarkable decrease in temperature over multiple successive temperature measurement scans. A possible explanation for this was given, but so far we have not been able to provide empirical founding for it. This provides another outlook for future research.
Appendices

A Image capture and processing software

In order to perform actual temperature measurements on the basis of images, and to improve the general workflow of the experimental setup, an advanced frontend for the CCD camera has been developed. Because its extensive (real-time) image processing and simple image acquisition interface we chose to use MATLAB 2008a for its implementation.

This appendix contains the full source code for the MATLAB program as well as a screenshot of the GUI with their internal object names superimposed and an illustration showing the functional relations within a program. All this information is provided both for reference and archival purposes.

As this software is released under the GNU General Public License, anyone is free to use, modify and distribute it given it be done under the exact same licensing conditions.
A.1 GUI

Figure 15: Screenshot of the GUI running with an actual MOT, superimposed with labels stating the component handler names as referred to in the MATLAB code and the function call schematic in Figure 16.
A.2 Function reference scheme

![Diagram showing function reference scheme](image)

Figure 16: Schematic display of GUI components (within the grey boxes) and ordinary function calls. The solid lines represent ordinary function calls, dotted lines signify a GUI object’s value being queried in that function and dashed lines signify callbacks.
A.3 Code

Listing 1: main.m - The code controlling the MATLAB GUI and image processing. Here "..." signifies a typographically enforced line-ending.

```matlab
function startCapture(hObject, handles)
    display('Capturing.')
    set(handles.captureshadow_button, 'Enable', 'on');
    set(handles.edit_button, 'Enable', 'on');
    set(handles.exposuretime, 'Enable', 'off');
    set(handles.mot_history_button, 'Enable', 'on');
    set(handles.image_history_button, 'Enable', 'on');
    set(handles.trigger_int, 'Enable', 'off');
    set(handles.trigger_ext, 'Enable', 'off');
    set(handles.log_data, 'Enable', 'off');
    set(handles.log_images, 'Enable', 'off');
    colormap jet;
    imaqreset;
    % Init video memory, limit to 25% of available memory
    available = imaqmem('AvailPhys');
    imaqmem(0.25*available);
    % Init video
    vid = videoinput('dcam',1,'F7Y16_752x480');
    % Setup capturing for one frame per trigger
    if get(handles.trigger_ext, 'Value') == get(handles.trigger_ext, 'Max')
        triggerconfig(vid, 'hardware', 'risingEdge', 'externalTrigger');
    else
        triggerconfig(vid, 'manual');
    end
    % Capture properties
    source = getselectedsource(vid);
    set(vid,'FramesPerTrigger', 1,'TriggerRepeat',Inf);
    set(vid,'FramesAcquiredFcnCount', 1);
    set(vid,'FramesAcquiredFcn', {[capture_callback(hObject, handles)});
    % Shutter time should be dynamically set from GUI
    shut = str2double(get(handles.exposuretime, 'String'));
    set(source,'ShutterAbsolute', 1e-3*shut,'Gain',0,'Brightness',0,'Gamma',0);
    set(handles.main_figure, 'UserData', vid);
    start(vid);
```
trigger(vid);

function stop(capture(hObject, handles)
    display('Not capturing.')
    try
        vid = get(handles.mainfigure, 'UserData');
        % Cleanup for vid
        stop(vid);
        delete(vid);
    catch
        % Do bogus
    end
    % Delete all images from memory
    delete(imaqfind);
    set(handles.capturebackgroundbutton, 'Enable', 'off');
    set(handles.exposure_time, 'Enable', 'on');
    set(handles.editbutton, 'Enable', 'off');
    % Re-enable trigger selection
    set(handles.trigger_int, 'Enable', 'on');
    set(handles.trigger_ext, 'Enable', 'on');
    % Re-enable logging buttons
    set(handles.logdata, 'Enable', 'on');
    set(handles.logimages, 'Enable', 'on');
    % Disable logging
    set(handles.logdata, 'Value', get(handles.logdata, 'Min'));
    set(handles.logimages, 'Value', get(handles.logimages, 'Min'));
    function capture(callback(object, event, hObject, handles)
        display('Caught a frame.');
        % Grab the image
        [image, time, metadata] = getdata(object, 1);
        capturebutton = handles.capturebackgroundbutton;
        substractbutton = handles.substractbackgroundbutton;
        editbutton = handles.editbutton;
        if get(capturebutton, 'UserData') == 1.0
            display('Grabbing a background, momma.');
            % Change this to the current image later
            set(substractbutton, 'UserData', image);
            set(substractbutton, 'Enable', 'on');
            set(capturebutton, 'UserData', 0.0);
            image = process_image(image);
        else
            if get(substractbutton, 'Value') == get(substractbutton, 'Max')
                display('Subtracting background.');
                image = process_image(image, get(substractbutton, 'UserData'));
            else
                image = process_image(image);
            end
        end
        display.image(hObject, handles, image);
        if get(editbutton, 'UserData') == 1.0
            set(editbutton, 'UserData', 0.0);
            imtool(image, [0 1023]);
        end
        clear image;
        if get(handles.trigger_ext, 'Value') == get(handles.trigger_ext, 'Min')
% We need to catch errors here - sometimes it goes wrong
try
  trigger(object);
catch
end
end

% This is the Gauss we do fitting to
% The parameters are:
% b(1): Scaling factor of Gauss
% b(2): Sigma
% b(3): Mu
% b(4): Constant background
function yhat = gauss(b, X)
  factor = b(1)/(b(2) * sqrt(2*pi));
  yhat = factor * exp(-((X-b(3))^2)/(2*b(2)^2)) + b(4);
end

function display_image(hObject, handles, image)
display('Display image');
% Parameters for reference drawing
xpos1 = 0.50;
ypos1 = 0.48;
xpos2 = 0.44;
ypos2 = 0.44;
xpos3 = 0.50;%(xpos1 + xpos2)/2;
ypos3 = 0.46;%(ypos1 + ypos2)/2;
width1 = .91;
width2 = .81;
width3 = 0.05;
% Screen updating feedback
upd_string = get(handles.updating, 'String');
if upd_string == '|'|
  set(handles.updating, 'String', '/|');
elseif upd_string == '/|
  set(handles.updating, 'String', '-|');
elseif upd_string == '-|
  set(handles.updating, 'String', '\-');
else
  set(handles.updating, 'String', '|\');
end
% Do zoom
if get(handles.zoom_switch, 'Value') >= get(handles.zoom_switch, 'Max')
  xmin = str2double(get(handles.zoom_range_start, 'String')) + 1;
xmax = str2double(get(handles.zoom_range_end, 'String')) + 1;
xrange = xmax - xmin;
scale = xrange/751;
yrange = round(scale*479);
ymin = round((1 - get(handles.zoom_slider, 'Value') - 479 - yrange)) + ... 1;
ymax = ymin + yrange;
image = image(ymin:ymax, xmin:xmax);
end
% Horizontal histogram
horizontal = transpose(sum(image, 2));
area(handles.histogram_axes, horizontal);

% Plot the fitted Gauss distribution over the original graph
% These initial values should have reasonable defaults
% Allow the possibility for 'smart' initial values, based on a fit.
% NOTE: this usually doesn't improve
[mot_mu, mot_sigma, mot_muci, mot_sigmaci] = normfit(mot_list, [], [],... horizontal);
% initial = [10^8 mot. sigma mot. mu 10^4];

% Sensible initial = [10^8 200 numel(horizontal)/2 10^4];

[beta, resid, J, sigma] = nlinfit(mot_list, horizontal, @gauss, initial);
betaci = nlinparci(beta, resid, 'covar', sigma);

% This code will not win beauty contests.

mot_factor = abs(beta(1));
mot_sigma = abs(beta(2));
mot_mu = abs(beta(3));
mot_factorci = abs(betaci(1,:));
mot_sigmaci = abs(betaci(2,:));
mot_muci = abs(betaci(3,:));

set(handles.histogram_axes, 'NextPlot', 'add');
plot(handles.histogram_axes, mot_list, gauss(beta, mot_list), 'r', 'LineWidth', 2);
set(handles.histogram_axes, 'NextPlot', 'replace');

else

mot_factor = zeros(1,1);
mot_sigma = zeros(1,1);
mot_mu = zeros(1,1);
mot_factorci = zeros(1,2);
mot_sigmaci = zeros(1,2);
mot_muci = zeros(1,2);
end

mot_mu_dev = 2*max(mot_mu - mot_muci(1), mot_muci(2) - mot_mu);
mot_sigma_dev = 2*max(mot_sigma - mot_sigmaci(1), mot_sigmaci(2) - ...
mot_sigma);
mot_factor_dev = 2*max(mot_factor - mot_factorci(1), mot_factorci(2) - ...
mot_factor);

set(handles.mot_mu, 'String', [num2str(mot_mu,'%3.2f') ' ± ' num2str(...
mot_mu_dev, '%2.1f')]);
set(handles.mot_mu, 'UserData', [mot_mu mot_muci(1) mot_muci(2)]);

set(handles.mot_sigma, 'String', [num2str(mot_sigma,'%3.2f') ' ± ' num2str(...
mot_sigmaci(1) mot_sigmaci(2) ...]

set(handles.mot_factor, 'String', [num2str(mot_factor/10^5, '%2.1f') 'e+5 ± ' num2str(mot_factorci(1) mot_factorci(2) ...]

image_mu = mean(image(:));
image_diff = image(:) - image_mu;
image_sigma = sqrt(sum(image_diff.*image_diff) / numel(image));

set(handles.image_mu, 'String', num2str(image_mu,'%3.2f'));
set(handles.image_mu, 'UserData', image_mu);

set(handles.image_sigma, 'String', num2str(image_sigma,'%3.2f'));
set(handles.image_sigma, 'UserData', image_sigma);

imagesc(image, 'Parent', handles.image_axes);

set(handles.image_extremes, 'String', [num2str(min(image(:))) ' − ' num2str(...
max(image(:))]);

% Image contrast scaling
if get(handles.levels_switch, 'Value') == get(handles.levels_switch, 'Max')
caxis(handles.image_axes, 'Max')
caxis(handles.image_axes, str2double(get(handles.levels_range_start, 'String'))) str2double(get(handles.levels_range_end, 'String')));
else

caxis(handles.image_axes, 0 1024)])
end

% Only display 'crop marks' when not zooming in
if ~get(handles.zoom_switch, 'Value') == get(handles.zoom_switch, 'Max'))
    rectangle('Position', [(xpos1 - width1/2)*752, (ypos1 - width1.../2+752/480)*480, width1+480*752/480], 'Curvature', [1,...1], 'LineWidth', 1.0, 'Parent', handles.image_axes);
end
if get(handles.history_logging, 'Value') == get(handles.history_logging, 'Max')
    histsize = str2double(get(handles.history_datapoints, 'String'));
    auto_range = ~get(handles.history_freeze, 'Value'));
if get(handles.history_logging, 'Value') == get(handles.history_logging, 'Max')
    histsize = str2double(get(handles.history_datapoints, 'String'));
    auto_range = ~get(handles.history_freeze, 'Value'));
end
histsize = str2double(get(handles.history_datapoints, 'String'));
auto_range = ~get(handles.history_freeze, 'Value'));
end
history_graph(graphHandles.mot_sum_axes, handles.mot_sum, histsize, ... auto_range);
history_graph(graphHandles.mot_sigma_axes, handles.mot_sigma, histsize,... auto_range);
graphHandles = guidata(handles.image_graphs);
history_graph(graphHandles.image_mu_axes, handles.image_mu, histsize, ... auto_range);
history_graph(graphHandles.image_sigma_axes, handles.image_sigma,... histsize, auto_range);
end
save_data(handles);
save_image(handles, image);
function [data_min, data_max] = find_extremes(data)
newdata = [];
for k = 1:length(data)
    if ~(data(k) == zeros(1,1));
        newdata = [newdata data(k)];
    end
end
data_max = max(newdata);
data_min = min(newdata);
function history_graph(graph_axes, data, histsize, auto_range)
% Sum history
userdata = get(graph_axes, 'UserData');
data = get(data, 'UserData');
if numel(userdata) == 0
    userdata = struct('val', zeros(1,histsize), 'min', zeros(1, histsize), ... 'max', zeros(1, histsize));
end
userdata.val = [userdata.val(2:end) data(1)];
if numel(data) == 3
    userdata.min = [userdata.min(2:end) data(2)];
    userdata.max = [ userdata.max(2:end) data(3)];
end
old_lim = ylim(graph_axes);
if numel(data) == 3
    plot(graph_axes, userdata.max, 'Color', 'green');
    set(graph_axes, 'NextPlot', 'add');
    plot(graph_axes, userdata.min, 'Color', 'green');
end
plot(graph_axes, userdata.val, 'k');
set(graph_axes, 'NextPlot', 'replace');
[data_min, data_max] = find_extremes(userdata.val);
if data_max == data_min
    data_margin = 0.2*data_max;
else
    data_margin = 0.2*(data_max - data_min);
end
if auto_range
    ylim(graph_axes, [data_min-data_margin data_max+data_margin]);
else
    ylim(graph_axes, old_lim);
end
line([0 histsize], [data_max data_max], 'Color', 'red', 'Parent', graph_axes);
line([0 histsize], [data_min data_min], 'Color', 'blue', 'Parent', ...
    graph_axes);
set(graph_axes, 'UserData', userdata);

function new_image = process_image(image, varargin)
% expects arguments (image, background) or (image)
if size(varargin, 2)
    image = imabsdiff(varargin{1}, image);
end
new_image = image;

function save_data(handles)
if get(handles.log_data, 'Value') == get(handles.log_data, 'Max')
    try
        filename = get(handles.log_data, 'UserDate');
        mot_sigma = get(handles.mot_sigma, 'UserDate');
        mot_mu = get(handles.mot_mu, 'UserDate');
        image_sigma = get(handles.image_sigma, 'UserDate');
        image_mu = get(handles.image_mu, 'UserDate');
        vid = get(handles.main_figure, 'UserData');
        framenumber = get(vid, 'FramesAcquired');
        data = [framenumber mot_sigma mot_mu image_sigma image_mu];
        ['Saving data to ' filename]
        dlmwrite(filename, data, '-append', 'delimiter', ',')
    catch ME
        'Not saving data.'
        rethrow(ME);
    end
end

function save_image(handles, image)
% Save current image
if get(handles.log_images, 'Value') == get(handles.log_images, 'Max')
    pathname = get(handles.log_images, 'UserData');
    try
        vid = get(handles.main_figure, 'UserData');
        framenumber = get(vid, 'FramesAcquired');
        % For 16-bit tiff files, uncomment this
        %filename = [pathname int2str(framenumber) '.tif'];
        % ['Saving image to ' filename]
        imwrite(image, pathname, 'tif', 'Compression', 'packbits')
        %
        %['Saving image to ' filename]
        %imwrite(image, pathname, 'jpg', 'Quality', 95);
    end
end
catch ME
    'Not saving image.'
    rethrow(ME);
end
end

function defaultpath = get_default_path(handles)
    basepath = '\Wtcw-server\qgases\groups\rydberg\matlab dataverwerking\...
               mathijs\data\';
    defaultpath = [basepath datestr(now, 29) '\'];
    mkdir(defaultpath);
end

function defaultfile = get_default_file(handles)
    defaultfile = [datestr(now,'HHMM') ' − ' get(handles.exposure_time, 'String...
               ')] 'ms'];
B  Timing and control software

To be able to precisely and repeatedly program and control the timing of this experiment I have written a (nearly) complete Python wrapper around the GPIB functionality of the used DG535 Digital Delay Generator.

Using this wrapper experimental control was easily implemented with a small and easily readable Python program, which is first. After this a full listing of the GPIB wrapper is given.

As the GPIB wrapper is released under the GNU General Public License, anyone is free to use, modify and distribute it given it be done under the exact same licensing conditions.

B.1  Experimental control program

Listing 2: experiment.py – Python code used for controlling the actual experiment. Here “...” signifies a typographically enforced line-ending.

```python
""" This experiment is made for scanning over the decay of an atom cloud. For this to work the following connections have to be made:
T0      =>   CCD trigger
A       =>   magnet switch (switches off magnets for 20ms)
AB, CD  =>   AOM video input

For used values and timing, see the code around line 56.
"""

from delaygenerator import *
from time import sleep
import winsound

def frange(start, end=None, inc=None):
    "A range function, that does accept float increments..."
    if end == None:
        end = start + 0.0
        start = 0.0
    if inc == None:
        inc = 1.0
    L = []
    while 1:
        next = start + len(L) * inc
        if inc > 0 and next >= end:
            break
        elif inc < 0 and next <= end:
            break
        L.append(next)
    return L

dg = DelayGenerator("GPIB::15")
dg.clear()

def message(msg):
    print msg
dg.set_display(msg)
message("Starting experiment.")
```

34
B.2 GPIB wrapper for DG535 Delay Generator

Listing 3: delaygenerator.py – Python wrapper around the GPIB command set to the DG535 Digital Delay Generator. Here “…” signifies a typographically enforced line-ending.
from visa import GpibInstrument

# Available channels
TRG=0
T0=1
A=2
B=3
AB=4
C=5
D=6
CD=7

# Available impedances
LOWIMP=0
HIGHIMP=1

# Available output modes
TTL=0
NIM=1
ECL=2
VAR=3

# Available output polarities
INV=0
NORM=1

# Available trigger modes
INT=0
EXT=1
SS=2
BURST=3

# Available trigger rates
INT_RATE=0
BURST_RATE=1

# Possible error values for error byte
UNR_COMMAND = int('00000001',2)
WRONG_PARAM = int('00000010',2)
OUT_RANGE = int('00000100',2)
WRONG_MODE = int('00001000',2)
DELAY_LINK = int('00010000',2)
DELAY_RANGE = int('01000000',2)
CORRUPT_DATA= int('01000000',2)

def check_channel(channel):
    """ Check whether this is a valid channel.
    Allowed values are: [TRG, T0, A, B, C, D, AB, CD]
    """
    assert channel in range(0, 8), 'Wrong channel specified: %s' % channel

def check_mode(mode):
    """ Check whether this is a valid mode. """
    assert mode in range(0, 4), 'Wronge mode specified: %s' % mode

class DelayError(Exception):
    """ Error class, a parsing wrapper around the error byte of the Delay ... Generator. """
    def __init__(self, byte):
        assert hasattr(byte, '__int__'), 'Error value not an integer.'
        self.byte = byte

    def __str__(self):
        errlist = []
        if self.byte & UNR_COMMAND:
            errlist.append('Unrecognized command.')
        if self.byte & WRONG_PARAM:
            errlist.append('Wrong number of parameters.')
if self.byte & OUT_RANGE:
    errlist.append('Value is outside allowed range.')
if self.byte & WRONG_MODE:
    errlist.append('Wrong mode for the command.')
if self.byte & DELAY_LINK:
    errlist.append('Delay linkage error.')
if self.byte & DELAY_RANGE:
    errlist.append('Delay range error.')
if self.byte & CORRUPT_DATA:
    errlist.append('Recalled data corrupt.')

if errlist:
    import string
    return string.join(errlist, ' ')
else:
    return 'An unknown error has occurred. Error byte: %b' % self.byte

class DelayGenerator(GpibInstrument):
    ""
    This is a wrapper class around the DG535 Digital Delay Generator.
    
    When using this one, make sure you import everything from this module
    because the constants defined here are required as well.
    
    So:
    from delaygenerator import *
    ""

def __init__(self, resource_name, **keyw):
    ""
    Instantiate a Delay Generator object.
    
    Parameters:
    - resource_name : The name of the GPIB resource to connect to.
    - delay : The delay to issue after sending a command. (Optional.)
    
    Example:
    dg = DelayGenerator("GPIB::15")
    ""

    if not keyw.has_key('delay'):
        keyw.update({'delay':0.1})
    super(GpibInstrument, self).__init__(resource_name, **keyw)

def command(self, message):
    ""
    Send a command to the Delay Generator and check for errors upon
    execution.
    
    This raises a DelayError when something goes wrong.
    ""

    self.write(message+'\n')
    self.get_error_status()

def get_error_status(self):
    ""
    Check the error byte of the DelayGenerator and raise a
    DelayError when one is found. ""

    super(GpibInstrument, self).write('ES')
    status = self.read()
    try:
        error_status = int(status)
    except:
        print 'Error parsing error status: %s' % error_status
    if error_status:
        raise DelayError(error_status)

def set_delay(self, ref, channel, time):
    ""
    Set the delay time of channel with respect to ref.
    ""

    Parameters:
    - ref : Reference channel for the delay.
    - channel : The channel for which to set the delay.
− time : Time in seconds between ref and channel.

Example:
dg.set_delay(A, D, 0.002)

assert time <= 1.0, 'Time should be 1.0 s or less.'
check_channel(channel)
check_channel(ref)
sel.command('DT%(channel)s,%(ref)s,%(time)f' % locals())

def set_display(self, message):
    """ Sets the DelayGenerator's display to message. """
    self.command('DS%s' % message.replace(' ', '_'))

def set_impedance(self, channel, impedance):
    """ Sets the output impedance of channels. """

    Parameters:
    - channel : The channel for which to set the impedance.
    - impedance : The impedance, either LOWIMP for 50 Ohm or
                  HIGHIMP for high impedance (~1 MOhm).
    
    check_channel(channel)

    assert impedance in [LOWIMP, HIGHIMP], 'Invalid impedance.'

    self.command('TZ%(channel)s,%(impedance)s' % locals())

def get_impedance(self, channel):
    """ Gets the impedance for channel. """
    check_channel(channel)

    self.ask('TZ%(channel)s' % locals())

def get_output(self, channel):
    """ Gets the output settings for channel as a dictionary
    with amplitude and offset for variable output or polarity
    for TTL. """

    check_channel(channel)

    mode = int(self.ask('OM%(channel)s' % locals()))

    if mode == VAR:
        amplitude = float(self.ask('OA%(channel)s' % locals()))
        offset = float(self.ask('OO%(channel)s' % locals()))
    else:
        polarity = int(self.ask('OP%(channel)s' % locals()))

    return locals()

def set_output(self, channel, mode, polarity=NORM, amplitude=None, offset=0.0):
    """ Sets the output for channel. """

    Parameters:
    - channel : The channel for which to set the output. If ...
                 AB or CD, VAR is the only allowed mode. ...  
                 Apparently.
    - mode : Output mode. VAR for variable voltage, TTL for...
            ... TTL.
    - polarity : When using TTL this sets the polarity to NORM ...
                 or INV (inverted).
    - amplitude : The amplitude of the output in Volt for mode=VAR...
    - offset : Offset of the output for mode=VAR.

    Example:
dg.set_output(AB, VAR, amplitude=amp, offset=off)
def set_output(T5, TTL, polarity=NORM):
    ***
    check_channel(channel)
    check_mode(mode)

    assert polarity in [INV, NORM], 'Invalid polarity given.'
    self.command('OM%(channel)s,%(mode)s' % locals())

    if mode == VAR:
        assert amplitude._float_, 'Output mode variable but amplitude not...
        set.'
        assert offset._float_, 'Offset should have a floating point value...
        '  
        assert amplitude+offset <= 4.0, 'Output voltage too high.'
        assert amplitude+offset >= -3.0, 'Output voltage too low.'
        self.command('OA%(channel)s,%(amplitude)f' % locals())
        self.command('OO%(channel)s,%(offset)f' % locals())
    else:
        assert channel != AB, 'AB only supports VAR.'
        assert channel != CD, 'CD only supports VAR.'
        self.command('OP%(channel)s,%(polarity)s' % locals())

    def set_trigger_mode(self, mode):
        *** Sets the trigger mode.

        Possible modes are:
        INT : Internal
        EXT : External
        SS : Single shot
        BURST : Burst

        ***
        check_mode(mode)

        self.command('TM%(mode)s' % locals())

    def set_trigger_rate(self, trigger_rate, rate):
        *** Sets the trigger rate for internal and burst triggering.

        Parameters:
        - trigger_rate : The type of rate to set.
        Options: INT_RATE, BURST_RATE
        - rate : The rate to set the triggering to in Hz.

        ***
        assert trigger_rate in [0, 1]
        assert rate, 'Rate not set.'
        assert rate >= pow(10, -3), 'Rate too low.'
        assert rate <= pow(10, 3), 'Rate too high.'
        self.command('TR%(trigger_rate)s,%(rate)f' % locals())

    def single_shot(self):
        *** Fire a single shot for the SS triggering mode."
        self.command('SS')


39
C GNU General Public License

Version 3, 29 June 2007


Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers’ and authors’ protection, the GPL clearly explains that there is no warranty for this free software. For both users’ and authors’ sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users’ freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for
those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

**TERMS AND CONDITIONS**

0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.


The “source code” for a work means the preferred form of the work for
making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms
that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.


No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work’s users, your or third parties’ legal rights to forbid circumvention of technological measures.


You may convey verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

(a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

(b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”.

(c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

(d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive inter-
faces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

(a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

(b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

(c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

(d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

(e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source
of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this Li-
cense by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

(a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

(b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

(c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

(d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

(e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

(f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form
of a separately written license, or stated as exceptions; the above require-
ments apply either way.

8. Termination.
You may not propagate or modify a covered work except as expressly pro-
vided under this License. Any attempt otherwise to propagate or modify
it is void, and will automatically terminate your rights under this License
(including any patent licenses granted under the third paragraph of section
11).

However, if you cease all violation of this License, then your license from a
particular copyright holder is reinstated (a) provisionally, unless and until
the copyright holder explicitly and finally terminates your license, and (b)
permanently, if the copyright holder fails to notify you of the violation by
some reasonable means prior to 60 days after the cessation.
Moreover, your license from a particular copyright holder is reinstated
permanently if the copyright holder notifies you of the violation by some
reasonable means, this is the first time you have received notice of violation
of this License (for any work) from that copyright holder, and you cure
the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the li-
censes of parties who have received copies or rights from you under this
License. If your rights have been terminated and not permanently rein-
stated, you do not qualify to receive new licenses for the same material
under section 10.

You are not required to accept this License in order to receive or run a
copy of the Program. Ancillary propagation of a covered work occurring
solely as a consequence of using peer-to-peer transmission to receive a
copy likewise does not require acceptance. However, nothing other than
this License grants you permission to propagate or modify any covered
work. These actions infringe copyright if you do not accept this License.
Therefore, by modifying or propagating a covered work, you indicate your
acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.
Each time you convey a covered work, the recipient automatically receives
a license from the original licensors, to run, modify and propagate that
work, subject to this License. You are not responsible for enforcing com-
pliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organi-
ization, or substantially all assets of one, or subdividing an organization,
or merging organizations. If propagation of a covered work results from
an entity transaction, each party to that transaction who receives a copy
of the work also receives whatever licenses to the work the party’s prede-
cessor in interest had or could give under the previous paragraph, plus a
right to possession of the Corresponding Source of the work from the pre-
decessor in interest, if the predecessor has it or can get it with reasonable
efforts.
You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.
A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007. Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.
Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.
The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.
If the Program specifies that a proxy can decide which future versions of
the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.
To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <textyear> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail. If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands show w and show c should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see http://www.gnu.org/licenses/.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read http://www.gnu.org/philosophy/why-not-lgpl.html.
References


