# Motional Quantum Ground-State Cooling Outside the Lamb-Dicke Regime – Supplemental material

Yichao Yu,* Nicholas R. Hutzler,† Jessie T. Zhang, Lee R. Liu, and Kang-Kuen Ni‡
*Department of Chemistry and Chemical Biology, Harvard University, Cambridge, Massachusetts, 02138, USA*
*Department of Physics, Harvard University, Cambridge, Massachusetts, 02138, USA and Harvard-MIT Center for Ultracold Atoms, Cambridge, Massachusetts, 02138, USA*
(Dated: August 1, 2017)

## 1. SIMULATION OF 3D RAMAN SIDEBAND COOLING

We use a computer simulation to study the importance of different effects and guide the optimization of Raman sideband cooling. The simulation uses the quantum jump (or Monte-Carlo wavefunction) method [1, 2] to accurately calculate the evolution within each laser pulse where different energy eigenstates are assumed to decohere immediately upon spontaneous emission and on pulse boundaries. Such approximations are valid and greatly simplify the computation at the cost of limiting the applicability of the simulation to processes that require stronger coherence, such as Ramsey spectroscopy or rapid push-out of the atoms from the trap (faster than a trap oscillation period).

A number of other simplifying approximations, listed below, are also used in the simulation. These either have a demonstrably negligible effect on the experiment, or result in a significant reduction of complexity for an effect that can be optimized more easily in the experiment.

1. *All Raman pulses resonantly drive only one sideband/carrier order.* We tweak the pulse shape, power and length as discussed in the text, for example by using Blackman pulses, to reduce coupling to other orders. We do see deviation from this approximation in the experiment, though mostly in the long detection pulses, and it is less of a concern for the short cooling pulses.

2. *No heating from other sources.*
   We measured the heating rate and found that it results mainly from off-resonant scattering of photons, which is included in the simulation. In particular, we can ignore heating from the switching trap.

3. *Same trapping frequency for all states.*
   This effect is negligible for Na in the ground state due to the small fine structure splitting in the excited state. It is straightforward to include this effect if the calculation needs to be repeated on a system where this is not the case (e.g. directly laser cooled molecules).

Some effects that are important to include are all three dimensions (3D motional states, accurate 3D wavevector, and different emission pattern for different photon polarizations), off-resonant scattering from all laser beams, and optical pumping polarization imperfection. The optical pumping polarization and off-resonant scattering are especially important for accurate simulation since they are the main limitation on cooling performance.

In general, each pulse has a coherent part from the Raman drive (which is absent for the optical pumping pulses) and an incoherent part from scattering (optical pumping or off-resonant). This nicely fits the general approach of the quantum jump method. Due to the large number of states that we need to take into account ($30 - 150$ motional per axis and 8 hyperfine, giving a total of $10^6$), generic wavefunction propagation is very inefficient. Therefore, we further simplify by reducing the number of states tracked during the coherent propagation and applying quantum jumps using a set of reduced jump operators that describe the total jump probability from a certain state (summed over all final states). The coherent propagation is then accurately solved in order to avoid having to use a time step much shorter than the inverse of scattering rate (up to hundreds of kHz)[4].

In order to simulate a pulse, we first start with the coherent propagation that includes the modification from the jump operators as in the quantum jump method. When a jump happens, we pick a final internal state to jump to

*Electronic address: yichaoyu@g.harvard.edu
†Present address: California Institute of Technology, Division of Physics, Mathematics, and Astronomy. Pasadena, CA, 91125
‡Electronic address: ni@chemistry.harvard.edu

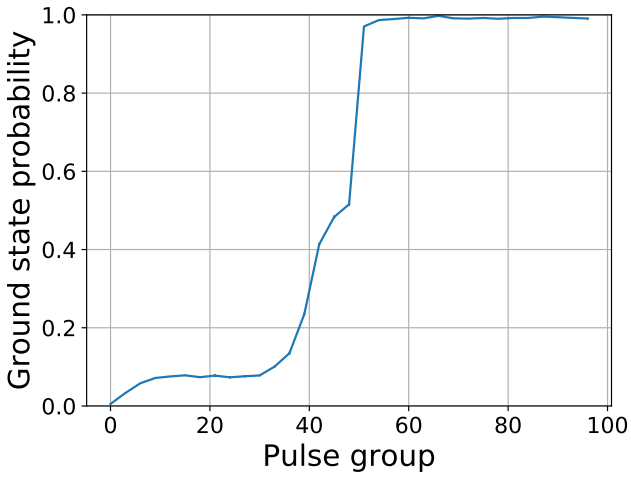


FIG. 1: Simulated 3D ground state population as a function of cooling cycles without technical imperfections. The ground state probability reaches $> 99\%$.

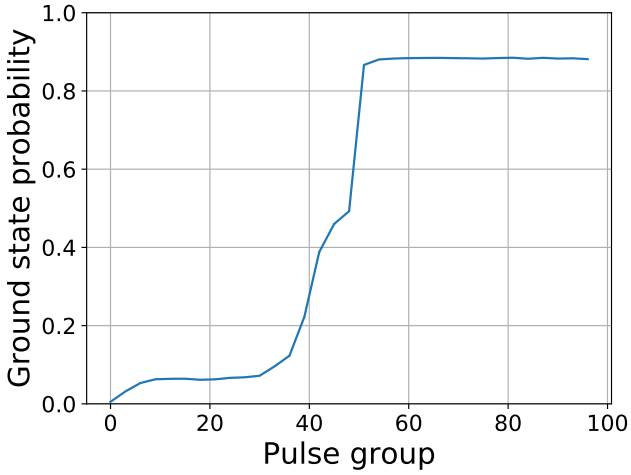


FIG. 2: Simulated 3D ground state population as a function of cooling cycles including effect of off-resonant scattering from all beams and OP misalignment. For this particular sequence, the ground state probability saturates to $\approx 88\%$.

based on the relative rates of different scattering sources and their branching ratios. This process determines the polarization and emission pattern of the emitted photon, from which we randomly sample a particular wave vector. The photon recoil being defined, we can then assign a final motional state to the atom. This process is then repeated as long as the pulse is on, before we start to simulate the next pulse.

We apply "virtual" measurements on the results of each simulated cooling sequence which we then average to obtain the expected value of experimental quantities. Due to the decoherence approximation, the measurement has to commute with the Hamiltonian. Particularly useful measurements used in our simulation are hyperfine state distribution, average motional energy, loss rate (where an atom is assumed to be lost if its motional energy goes above a certain threshold at any time) probability of being in certain states (e.g. ground state).

If off-resonant scattering and optical pumping polarization misalignment are ignored, the simulation shows that there is not a cooling limit despite the large Lamb-Dicke parameter and high initial temperature (Fig 1). However, when these effects are taken into account, the ground state probability can saturate at a lower value (Fig 2) and require much more careful tweaking. The change of slop in these plots corresponds to changing the sideband orders used for cooling.

The simulation is written in Julia[3]. All code is available under the GPLv3 license at [5] and [6]

## 2. COOLING SEQUENCE

```
function s = NaSingleAtom(dummy)

%% Do NOT keep a change log here. Do it properly with git commits.

% error('Check TTL Overrides')

% ImageOnly will skip everything except imaging steps. This includes Raman,
% OP, lowering, blasting, etc. Use to get baseline survival rates just from
% imaging.
ImageOnly = 0;

% AllowLowering will over-ride ImageOnly and allow the lowering of the
% trap. Blasting will still be over-ridden.
AllowLowering = 1;

%Verbose? Decides whether or not to display which sequences are being run.
Verbose = 0;

%% Template for combining multiple 1D scans together.
% Scans:
% 1. Radial 3 +1 Rabi
% 2. Radial 3 0 Rabi
% 3. Co-prop 2, -2 / 2, -1
% 4. Overall survival

% Params1 = 0:6:180;
% Params2 = 2:2:80;
% % Params3 = [linspace(-18.55, -18.45, 11), linspace(-12.3, -12.2, 11)];
% Params4 = 1;

% offset1 = length(Params1);
% offset2 = offset1 + length(Params2);
% offset3 = offset2 + length(Params3);
% offset4 = offset3 + length(Params4);
% offset5 = offset4 + length(Params5);
% offset6 = offset5 + length(Params6);
% offset7 = offset6 + length(Params7);
% offset8 = offset7 + length(Params8);
% offset9 = offset8 + length(Params9);
% offset10 = offset9 + length(Params10);
% offset11 = offset10 + length(Params11);
% offset12 = offset11 + length(Params12);
% offset13 = offset12 + length(Params13);
% offset14 = offset13 + length(Params14);
% offset15 = offset14 + length(Params15);
% offset16 = offset15 + length(Params16);

% if Idx <= offset1
%     % 1. Radial 3 +1 Rabi
%     TNaRaman1 = Params1(Idx) * 1e-6;
%     NaRaman1Det = -19.056e6;
%     AmpNaRaman1F1 = 0.22;
%     AmpNaRaman1F2 = 0.05;
%     NaRaman1F1Vertical = 0;
%     NaRaman1F2CounterOP = 1;
```

```
% elseif Idx <= offset2
%     % 2. Radial 3 0 Rabi
%     Idx = Idx - offset1;
%     TNaRaman1 = Params2(Idx) * 1e-6;
%     NaRaman1Det = -18.462e6;
%     AmpNaRaman1F1 = 0.22;
%     AmpNaRaman1F2 = 0.05;
%     NaRaman1F1Vertical = 0;
%     NaRaman1F2CounterOP = 1;
% elseif Idx <= offset3
%     % 3. Co-prop 2, -2 / 2, -1
%     Idx = Idx - offset2;
%     TNaRaman1 = 13.25e-6;
%     NaRaman1Det = Params3(Idx) * 1e6;
%     AmpNaRaman1F1 = 0.25;
%     AmpNaRaman1F2 = 0.22;
%     NaRaman1F1Vertical = 0;
%     NaRaman1F2CounterOP = 0;
% elseif Idx <= offset4
%     % 4. Overall survival
%     Idx = Idx - offset3;
%     TNaRaman1 = 0;
%     BlastType = -2;
% else
%     error('Idx too large')
% end

s = ExpSeq('NaSingleAtom');

%% Note about coding style
% * DO NOT EVER place all your variables at the beginning of the
%     function (or other scope) unless you are a computer or compiler.
%     Early declaration is easier for the compiler to reason about
%     but it absolutely provide no benefit for programmers. There are very
%     good reasons why C++ and C99 allow intermingled declarations and why
%     basically all high level programming languages following the same
%     model allow it.
%
%     Please put the definitions of the variables closer
%     to where you actually use them or at least closer to when it is
%     logically meaningful as part of the sequence. (e.g. you should put
%     the settings of a certian step right before the step is defined
%     rather than at the beginning of the function, before the sequence
%     is defined, or right before the first use of the value).
%
%     It is perfectly fine to put some global settings early in the
%     function, just think twice before doing that and absolutely DON'T do
%     this for the majority of the variables.

%% MOT + single loading + imaging parameters

CsLOFreq = @(det) (8.84101177 * 1e9 - 110e6 - det) / s.CsLOScale;

%% Tweezer parameters
VTweezerNa = 0.5; % 41mW into the objective
VTweezerOffset = -0.035;
VTweezerNaStart = VTweezerNa; %Start high, then ramp down to final value
ACTweezerDuringDrop  = 1; %Use DC tweezer while dropping MOT?
```

```
ACTweezerDuringImage = 1; %Use DC tweezer while imaging?

%% MOT parameters
LoadMOTOnly = 0; %Only load MOT, then stop
MOTPiezo = 1;
DetMOT = -8e6;
AmpNaResDPMOT = s.NaDPACAmp;
VMOT = -9;

%% MOT parameters that might need tweaking
TLoadMOT = 200e-3;
AmpMOT = 0.4; %5 mW from fiber with -10 MHz
[Vx, Vy, Vz] = deal(0.53, -0.51, 0.53);    %[0.513, -0.47, 0.5]

dVx = 0;
dVy = 0;
dVz = 0;
[Vx, Vy, Vz] = deal(Vx + dVx, Vy + dVy, Vz + dVz);

%% Na Atom loading parameters
LoadFromMolasses = 1;
TLoadAtom = 4e-3;
VShimZero = [0, -0.5, +0.2];
AmpLoadAtom = 0.4; % 5 mW from fiber with -20 MHz
DetAtomLoad = -20e6;
AmpNaResDPAtomLoad = s.NaDPACAmp;

%% Cs loading parameters
TLoadCsMOT = 100e-3;
CsDetMOT = -7e6;
CsAmpRPMOT = 0.43; %.255; %58 uW measured in same place
CsAmpMOT = 0.83;    %.305; %1.50 mW after cleanup cube on upper breadboard
CsVTweezer = 0.25;    %1.15;
CsAOBD0 = 48e6;


%% Imaging parameters
DetImage = -15e6;
AmpNaResDPImage = s.NaDPACAmp;
AmpNaImage  = 0.2;
%TImage = 0.65e-3; %Na imaging time
TImage = 10e-3; %Cs imaging time
TMOTRelease = 10e-3; 100e-3;
SecondImage = 1; % Single atom
ThirdImage = 1; % After holding atom
TThirdImageDelay = 18.5e-3;
CsDetImage = -7e6;
AmpNaRPDPImage = 1;


%% Sequence

if Verbose
    blurt = @(x) disp(x);
else
    blurt = @(x) [];
end
```

```
%% Init
s.addStep(1e-3) ...
    .add('TiSapph/AMP', 1) ...
    .add('TiSapph/FREQ', 80e6) ...
    .add('TTLpiezomirror', MOTPiezo) ...
    .add('NaF2DP110/FREQ', s.NaF2DP110Center - DetMOT / 2) ...
    .add('NaF2DP110/AMP', AmpNaResDPMOT) ...
    .add('NaMOT/AMP', AmpMOT) ...
    .add('NaMOT/FREQ', 120e6) ...
    .add('VBShimX', Vx) ... %more negative moves right)
    .add('VBShimY', Vy) ...
    .add('VBShimZ', Vz) ...
    .add('VMOTCur', VMOT) ...
    .add('TTLSwitchToggleTiSapph', 1) ...
    .add('TTLSwitchToggleNa', 1) ...
    .add('NaOP/AMP', 0) ...
    .add('NaOP/FREQ', 120e6) ...
    .add('NaRaman1/AMP', 0) ...
    .add('NaRaman2/AMP', 0) ...
    .add('NaRaman17/AMP', 0) ...
    .add('NaRaman1/FREQ', 75e6) ...
    .add('NaRaman2/FREQ', 75e6) ...
    .add('NaRaman17/FREQ', 835e6) ...
    .add('NaF1DP400/AMP', 1) ...
    .add('NaF1DP400/FREQ', 380e6) ...
    .add('NaF2DP400/AMP', 1) ...
    .add('NaF2DP400/FREQ', 365e6) ...
    .add('NaD1/AMP', 0) ...
    .add('NaD1/FREQ', 95e6) ...
    .add('NaRamanDP/AMP', 0.22) ...
    .add('VTweezerNa', VTweezerNaStart) ...
    .add('Cs_AOBD/FREQ', CsAOBD0) ...
    .add('Cs_AOBD/AMP', 1);
%% Cs
s.add('FreqCsLO', CsLOFreq(CsDetMOT));
s.add('AmpCsRepump', CsAmpRPMOT);
s.add('AmpCsCool', 0);
s.add('VTweezer', CsVTweezer);

s.wait(2e-3);

if LoadMOTOnly
    blurt('Only loading MOT, then aborting');
    s.run();
    return;
end

%% Dump MOT
% s.addStep(100e-3) ...
%     .add('NaMOT/AMP', 0);


%% MOT loading time
% Note that Na MOT is always loading.
% Setting TLoadCsMOT to 0 will disable Cs MOT loading.
TMaxLoad = max(TLoadMOT, TLoadCsMOT);
```

```
if TMaxLoad > 0
    s.wait(TMaxLoad);
    if TLoadMOT > 0
        s.addStep(-TLoadMOT) ...
            .add('NaMOT/AMP', AmpMOT);
    end
    if TLoadCsMOT > 0
        s.addStep(-TLoadCsMOT) ...
            .add('AmpCsCool', CsAmpMOT);
    end
end

TMOT2 = 3e-3;
if TMOT2 > 0
    %% CMOT Stage
    blurt('CMOT');
    AmpMOT2Start = 0.3;
    AmpMOT2 = 0.15;
    DetMOT2Start = -10e6;
    DetMOT2 = -10e6;
    s.addStep(TMOT2) ...
        .add('NaMOT/AMP', linearRamp(AmpMOT2Start, AmpMOT2)) ...
        .add('NaF2DP110/FREQ', linearRamp(s.NaF2DP110Center - DetMOT2Start / 2, s.NaF2DP11
end

s.addStep(3e-6) ...
    .add('AmpCsRepump', 0) ...
    .add('AmpCsCool', 0) ...
    .add('NaMOT/AMP', 0);

%% Atom loading step
if LoadFromMolasses
    blurt('Load atom from molasses');
    % TODO F1 amp
    s.addStep(@SetVBFieldSubSequence, 0, VShimZero, 300e-6);
    s.addStep(3e-6) ...
        .add('NaF2DP110/FREQ', s.NaF2DP110Center - DetAtomLoad / 2) ...
        .add('NaF2DP110/AMP', AmpNaResDPAtomLoad) ...
        .add('NaF1DP400/AMP', 1);
    s.addStep(TLoadAtom) ...
        .add('NaMOT/AMP', AmpLoadAtom);
    s.add('NaMOT/AMP', 0);
end

%% Lasers and B field off, wait for MOT to disperse
s.add('NaMOT/AMP', 0);
s.add('AmpCsCool', 0);
s.addStep(@SetVBFieldSubSequence, 0, VShimZero, 1e-3);

s.add('TTLSwitchToggleTiSapph', ACTweezerDuringDrop);
s.wait(TMOTRelease);

dumpTweezerAtStart = 0;
if dumpTweezerAtStart
    blurt('*DUMP TWEEZER AFTER LOADING*');
    %% Dump tweezer after loading.
    s.add('VTweezerNa',0);
    s.wait(1e-3);
```

```
    s.add('VTweezerNa', VTweezerNa);
end

%% Imaging parameters
CsAmpImage =.5;0.305;
CsAmpRPImage = .5;.255;
TImgPreTrigger = 2e-3;
TImgPostTrigger = 1.5e-3;
TImgPostOff = 5e-3;

if SecondImage
    %% Second image - single atom, hopefully
    blurt('Image');
    s.addStep(10e-6) ...
        .add('TTLSwitchToggleTiSapph', ACTweezerDuringImage) ...
        .add('NaF2DP110/FREQ', s.NaF2DP110Center - DetImage / 2) ...
        .add('NaF2DP110/AMP', AmpNaResDPImage) ...
        .add('NaF1DP400/AMP', AmpNaRPDPImage) ...
        .add('NaF2DP400/AMP', 1) ...
        .add('NaMOT/AMP', 0) ...
        .add('FreqCsLO', CsLOFreq(CsDetImage));
    s.wait(TImgPreTrigger);
    s.addStep(TImgPostTrigger) ...
        .add('TTLAndor',1);
    s.addStep(TImage) ...
        .add('NaMOT/AMP', AmpNaImage) ...
        .add('AmpCsRepump', CsAmpRPImage) ...
        .add('AmpCsCool', CsAmpImage);
    s.addStep(3e-6) ...
        .add('NaMOT/AMP', 0) ...
        .add('AmpCsRepump', 0) ...
        .add('AmpCsCool', 0);
    s.add('TTLpiezomirror', 0);
    s.addStep(TImgPostOff) ...
        .add('TTLAndor',0);
    % This is waited for before taken the next image
    s.addBackground(TThirdImageDelay);
end

%% Test trap merging
TMergeLower = 0;
if TMergeLower > 0
    VTweezerCsPreMerge = 0.9;
    TPreMergeLower = 5e-3;
    VTweezerNaPreMerge = 0.023;
    MergeImgCs = 0;
    dMerge = 2.5; % in um; how far away should I start Cs tweezer
    tmerge = 0.279986 * (abs(dMerge) * 1e-6)^(1/3);
    CsAOBD1 = CsAOBD0 + dMerge * 1e6 / 0.3125;

    % turn up Cs tweezer to RSC/merge value and turn down Na tweezer
    s.addStep(TPreMergeLower) ...
        .add('VTweezerNa', LowerAdiabatically(VTweezerNaPreMerge)) ...
        .add('VTweezer', LowerAdiabatically(VTweezerCsPreMerge));
    if tmerge > 0
        s.addStep(tmerge) ...
            .add('Cs_AOBD/FREQ', trapMove(CsAOBD0, CsAOBD1));
    end
```

```
    s.addStep(TMergeLower) ...
        .add('VTweezerNa', LowerAdiabatically(VTweezerOffset));
    % atoms are now in same tweezer; "image" them with Cs light
    if MergeImgCs
        s.addStep(20e-3) ...
            .add('AmpCsRepump', CsAmpRPImage) ...
            .add('AmpCsCool', CsAmpImage);
        s.addStep(1e-5) ...
            .add('AmpCsRepump', 0) ...
            .add('AmpCsCool', 0);
    end
    % raise 700nm tweezer again
    s.addStep(TMergeLower) ...
        .add('VTweezerNa', LowerAdiabatically(VTweezerNaPreMerge));
    if tmerge > 0
        s.addStep(tmerge) ...
            .add('Cs_AOBD/FREQ', trapMove(CsAOBD1, CsAOBD0));
    end
    % turn Cs tweezer back to load/image value and turn up Na tweezer
    s.addStep(TPreMergeLower) ...
        .add('VTweezerNa', LowerAdiabatically(VTweezerNa)) ...
        .add('VTweezer', LowerAdiabatically(CsVTweezer));

end

TCoolHold = 0; 0.2e-3;
DetCoolHold = -15e6;
AmpNaResDPICoolHold = s.NaDPACAmp;
VTweezerNaCoolHold = VTweezerNa;
AmpNaCoolHold = 0.23;
AmpNaF1DPCoolHold = 1;
if TCoolHold > 0
    %% Lifetime when cooling light is on.
    blurt('Measure Lifetime with cooling light on');
    s.addStep(10e-6) ...
        .add('TTLSwitchToggleTiSapph', 1) ...
        .add('NaF2DP110/FREQ', s.NaF2DP110Center - DetCoolHold / 2) ...
        .add('NaF2DP110/AMP', AmpNaResDPICoolHold) ...
        .add('NaF1DP400/AMP', AmpNaF1DPCoolHold) ...
        .add('NaF2DP400/AMP', 1) ...
        .add('NaMOT/AMP', 0);
    s.addStep(10e-3) ...
        .add('VTweezerNa', rampTo(VTweezerNaCoolHold));
    s.addStep(TCoolHold) ...
        .add('NaMOT/AMP', AmpNaCoolHold);
    s.addStep(10e-3) ...
        .add('NaMOT/AMP', 0) ...
        .add('VTweezerNa', rampTo(VTweezerNa));
    s.add('NaF1DP400/AMP', 1);
end

s.add('TTLpiezomirror', 0);
s.addStep(10e-6) ...
    .add('NaF2DP110/AMP', 0) ...
    .add('NaF1DP400/AMP', 0) ...
    .add('NaF2DP400/AMP', 0) ...
    .add('NaMOT/AMP', 0);
```

```
%% OP shim values
VShimOPX = -2.791;
VShimOPY = 0;
VShimOPZ = 0;
VShimOP = [VShimOPX VShimOPY VShimOPZ];
% Turn on a weak bias field along X direction for optical pumping
OPReverseB = 0;
if OPReverseB
    %% Reverse B_x first
    VShimActualOPX = VShimZero(1) - VShimOP(1);
    VShimActualOPY = VShimZero(2) + VShimOP(2);
    VShimActualOPZ = VShimZero(3) + VShimOP(3);
else
    %% Go to final field directly
    VShimActualOPX = VShimZero(1) + VShimOP(1);
    VShimActualOPY = VShimZero(2) + VShimOP(2);
    VShimActualOPZ = VShimZero(3) + VShimOP(3);
end
VShimActualOP = [VShimActualOPX VShimActualOPY VShimActualOPZ];
s.addStep(@SetVBFieldSubSequence, 0, VShimActualOP, 300e-6);
s.wait(5e-3); % wait for magnetic field to stabilize

TOPD1 = 0; 400e-6;
if (TOPD1 > 0) && ~ImageOnly
    %% D1 optical pumping into 2,-2
    AmpF1OP = 0.28;
    AmpF2D1OP = 0.2;
    s.addStep(1e-6) ...
        .add('NaOP/AMP', 0.25);
    s.addStep(TOPD1) ...
        .add('NaF1DP400/AMP', AmpF1OP) ...
        .add('NaD1/AMP', AmpF2D1OP);
    s.addStep(3e-6) ...
        .add('NaF1DP400/AMP', 0) ...
        .add('NaD1/AMP', 0) ...
        .add('NaOP/AMP', 0);
end

TDepumpD1 = 0; 7e-3;
if TDepumpD1 > 0
    %% Depumping
    AmpDepumpD1 = 0.12;
    s.addStep(TDepumpD1) ...
        .add('NaD1/AMP', AmpDepumpD1);
    s.addStep(1e-6) ...
        .add('NaD1/AMP', 0);
end

if OPReverseB
    %% adiabatically reverse magnetic field
    % Reverse B_x back
    s.addStep(300e-6) ...
        .add('VBShimX', rampTo(VShimZero(1))) ...
        .add('VBShimY', rampTo(VShimActualOPY + 2));
    s.addStep(300e-6) ...
        .add('VBShimX', rampTo(VShimZero(1) + VShimOP(1))) ...
        .add('VBShimY', rampTo(VShimActualOPY));
end
```

```
%%
VTweezerNaRaman = VTweezerNa;
if (VTweezerNaRaman ~= VTweezerNa) && ~ImageOnly
    s.addStep(10e-3)...
        .add('VTweezerNa', rampTo(VTweezerNaRaman));
end

%% Raman Cooling
SkipCool = 1;
state = NaRamanState(s);
state.skipCW = SkipCool;
state.skipPulsed = SkipCool;
% The following Pulsed cooling no-op
state.PulseOPF1 = 0.32;
state.TPulseOP = 40e-6;

% NaCWCool(T, Det, F1Vert, F2Counter, RamanF1, RamanF2, OPF1, OPF2)
% NaPulsedCool(T, Det, F1Vert, F2Counter, RamanF1, RamanF2, OPF2)
% NCoolCycles = 20;
NaAxial0 = -18.506e6;
NaAxialF = 67.5e3;

NaAxial8 = NaAxial0 + NaAxialF * 8;
NaAxial7 = NaAxial0 + NaAxialF * 7;
NaAxial6 = NaAxial0 + NaAxialF * 6;
NaAxial5 = NaAxial0 + NaAxialF * 5;
NaAxial4 = NaAxial0 + NaAxialF * 4;
NaAxial3 = NaAxial0 + NaAxialF * 3;
NaAxial2 = NaAxial0 + NaAxialF * 2;
NaAxial1 = NaAxial0 + NaAxialF;
NaAxial1_2 = NaAxial0 + 77e3;

TNaRaman1 = 0;
if TNaRaman1 < 0
    TNaRaman1 = -TNaRaman1;
    TN_1 = 1;
else
    TN_1 = 0;
end

NaCools0 = {NaPulsedCool(25e-6, -17.6375e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(25e-6, -18.03e6, 1, 1, 0.0, 0.05, 0.23, 1.0, 0.0), ...
            NaPulsedCool(25e-6, -17.2825e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0), ...
            NaPulsedCool(21e-6, -17.875e6, 0, 1, 0.0, 0.05, 0.23, 0.22, 0.0)};
NaCools0 = repmat(NaCools0, 1, 5);
NaCools1 = {NaPulsedCool(25e-6, -17.6375e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(25e-6, -18.03e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(25e-6, -17.2825e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0), ...
            NaPulsedCool(21e-6, -17.875e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0)};
NaCools1 = repmat(NaCools1, 1, 0);
% Axial 8+7, Radial2 2+1, Radial3 2+1
NaCools2 = {NaPulsedCool(80e-6 * TN_1, NaAxial8, 1, 0, 0.4, 0.14, 0.21), ...
            NaPulsedCool(80e-6, NaAxial7, 1, 0, 0.4, 0.14, 0.21), ...
            NaPulsedCool(25e-6, -17.6375e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(25e-6, -18.03e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(80e-6 * TN_1, NaAxial8, 1, 0, 0.4, 0.14, 0.21), ...
            NaPulsedCool(80e-6, NaAxial7, 1, 0, 0.4, 0.14, 0.21), ...
```

```
            NaPulsedCool(25e-6, -17.2825e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0), ...
            NaPulsedCool(21e-6, -17.875e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0)};
NaCools2 = repmat(NaCools2, 1, 5);
% Axial 7+6, Radial2 2+1, Radial3 2+1
NaCools3 = {NaPulsedCool(80e-6 * TN_1, NaAxial7, 1, 0, 0.4, 0.14, 0.21), ...
            NaPulsedCool(80e-6, NaAxial6, 1, 0, 0.4, 0.14, 0.21), ...
            NaPulsedCool(25e-6, -17.6375e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(25e-6, -18.03e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(80e-6 * TN_1, NaAxial7, 1, 0, 0.4, 0.14, 0.21), ...
            NaPulsedCool(80e-6, NaAxial6, 1, 0, 0.4, 0.14, 0.21), ...
            NaPulsedCool(25e-6, -17.2825e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0), ...
            NaPulsedCool(21e-6, -17.875e6, 0, 1, 0, 0.05, 0.21, 0.22, 0.0)};
NaCools3 = repmat(NaCools3, 1, 6);
% Axial 6+5, Radial2 2+1, Radial3 2+1
NaCools4 = {NaPulsedCool(80e-6 * TN_1, NaAxial6, 1, 0, 0.4, 0.14, 0.21), ...
            NaPulsedCool(60e-6, NaAxial5, 1, 0, 0.4, 0.14, 0.21), ...
            NaPulsedCool(25e-6, -17.6375e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(25e-6, -18.03e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(80e-6 * TN_1, NaAxial6, 1, 0, 0.4, 0.14, 0.21), ...
            NaPulsedCool(60e-6, NaAxial5, 1, 0, 0.4, 0.14, 0.21), ...
            NaPulsedCool(25e-6, -17.2825e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0), ...
            NaPulsedCool(21e-6, -17.875e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0)};
NaCools4 = repmat(NaCools4, 1, 7);
% Axial 5+4, Radial2 1, Radial3 1
NaCools5 = {NaPulsedCool(90e-6 * TN_1, NaAxial5, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
            NaPulsedCool(38e-6, -18.03e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(75e-6, NaAxial4, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
            NaPulsedCool(25e-6, -18.03e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(90e-6 * TN_1, NaAxial5, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
            NaPulsedCool(43e-6, -17.875e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0), ...
            NaPulsedCool(75e-6, NaAxial4, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
            NaPulsedCool(28e-6, -17.875e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0)};
NaCools5 = repmat(NaCools5, 1, 7);
% Axial 4+3, Radial2 1, Radial3 1
NaCools6 = {NaPulsedCool(75e-6 * TN_1, NaAxial4, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
            NaPulsedCool(38e-6, -18.03e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(60e-6, NaAxial3, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
            NaPulsedCool(25e-6, -18.03e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(75e-6 * TN_1, NaAxial4, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
            NaPulsedCool(43e-6, -17.875e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0), ...
            NaPulsedCool(60e-6, NaAxial3, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
            NaPulsedCool(28e-6, -17.875e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0)};
NaCools6 = repmat(NaCools6, 1, 8);
% Axial 3+2, Radial2 1, Radial3 1
NaCools7 = {NaPulsedCool(60e-6 * TN_1, NaAxial3, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
            NaPulsedCool(38e-6, -18.03e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(60e-6, NaAxial2, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
            NaPulsedCool(25e-6, -18.03e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(60e-6 * TN_1, NaAxial3, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
            NaPulsedCool(43e-6, -17.875e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0), ...
            NaPulsedCool(60e-6, NaAxial2, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
            NaPulsedCool(28e-6, -17.875e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0)};
NaCools7 = repmat(NaCools7, 1, 10);
% Axial 2+1, Radial2 1, Radial3 1
NaCools81 = {NaPulsedCool(60e-6 * TN_1, NaAxial2, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
             NaPulsedCool(38e-6, -18.03e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
             NaPulsedCool(60e-6, NaAxial1, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
             NaPulsedCool(25e-6, -18.03e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
```

```
            NaPulsedCool(60e-6 * TN_1, NaAxial2, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
            NaPulsedCool(43e-6, -17.875e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0), ...
            NaPulsedCool(60e-6, NaAxial1, 1, 0, 0.4, 0.00, 0.21, 0.0, 0.22), ...
            NaPulsedCool(28e-6, -17.875e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0)};
NaCools8Axial = NaPulsedCool(100e-6, NaAxial1_2, 1, 0, 0.2, 0.0, 0.21, 0.0, 0.22);
NaCools8 = {NaPulsedCool(70e-6, NaAxial1_2, 1, 0, 0.2, 0.0, 0.21, 0.0, 0.22), ...
            NaCools8Axial, ...
            NaPulsedCool(21e-6, -17.875e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0), ...
            NaPulsedCool(70e-6, NaAxial1_2, 1, 0, 0.2, 0.0, 0.21, 0.0, 0.22), ...
            NaCools8Axial, ...
            NaPulsedCool(25e-6, -18.030e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0), ...
            NaPulsedCool(70e-6, NaAxial1_2, 1, 0, 0.2, 0.0, 0.21, 0.0, 0.22), ...
            NaCools8Axial, ...
            NaPulsedCool(43e-6, -17.875e6, 0, 1, 0.0, 0.05, 0.21, 0.22, 0.0), ...
            NaPulsedCool(70e-6, NaAxial1_2, 1, 0, 0.2, 0.0, 0.21, 0.0, 0.22), ...
            NaCools8Axial, ...
            NaPulsedCool(38e-6, -18.030e6, 1, 1, 0.0, 0.05, 0.21, 1.0, 0.0)};
NaCools8 = [repmat(NaCools81, 1, 10), repmat(NaCools8, 1, 40)];
Type = 0;
if Type == 0
    NaCools = [NaCools0, NaCools1, NaCools2, NaCools3, NaCools4, ...
               NaCools5, NaCools6, NaCools7, NaCools6, NaCools7, NaCools8];
else
    NaCools = [NaCools8];
end
% NaCools = [NaCools0, NaCools1];
NaMaxCool = length(NaCools);
for i = 1:length(NaCools)
    if i > NaMaxCool
        break;
    end
    NaCools{i}.run(s, state);
end
if state.hasCool
    s.addStep(100e-6) ...
        .add('NaF1DP400/AMP', 0.32) ...
        .add('NaD1/AMP', 0.0) ...
        .add('NaOP/AMP', 0.25) ...
        .add('NaRaman1/AMP', 0) ...
        .add('NaRaman2/AMP', 0);
    s.addStep(5e-6) ...
        .add('NaF1DP400/AMP', 0) ...
        .add('NaOP/AMP', 0);
end

%% General purpose wait
TWait = 0; 2e-3;
if TWait > 0
    blurt(['Wait for ' int2str(TWait*1e3) ' ms']);
    % s.add('FPGA1/DDS6/AMP', 0);
    s.wait(TWait);
elseif TWait < 0
    s.addStep(-TWait) ...
        .add('FPGA1/DDS6/FREQ', 80e6) ...
        .add('FPGA1/DDS6/AMP', 0.3);
    s.add('FPGA1/DDS6/AMP', 0);
end
```

```
%% Spilling, i.e. inefficient evaporative cooling
DoSpill = 0;
if DoSpill && ~ImageOnly
    blurt('Spill atom');
    s.addStep(20e-3) ...
        .add('VTweezerNa', rampTo(0.2));
    s.wait(1e-3);
    s.addStep(20e-3) ...
        .add('VTweezerNa', rampTo(VTweezerNaRaman));
    s.wait(100e-6);
end

%% End of state preparation and beginning of detection

% TNaRaman1 = 100e-6;
if TNaRaman1 < 0
    TNaRaman1 = -TNaRaman1;
    error('TNaRaman1 < 0');
else
end
if (TNaRaman1 > 0) && ~ImageOnly
    %% Na Raman spectroscopy
    if TNaRaman1 > 1
        error('TNaRaman1 is too long!')
    end
    blurt('Do Raman Spectroscopy before lowering/blasting');
    NaRaman1Det = -18.489e6;
    NaRaman1Delta = 0; 20e3; % Doesn't need change
    AmpNaRaman1F1 = 0.2;
    AmpNaRaman1F2 = 0.1;
    AmpNaRaman1F1Ramp = 0.0;
    AmpNaRaman1F2Ramp = 0.0;
    NaRaman1F1Vertical = 1;
    NaRaman1F2CounterOP = 0;
    if abs(NaRaman1Det) < 1e6
        error('NaRaman1Det is too small!')
    end
    s.addStep(@NaRamanPrepStep, NaRaman1F1Vertical, NaRaman1F2CounterOP);
    s.addStep(@NaRamanStep, TNaRaman1, NaRaman1Det, NaRaman1Delta, ...
              AmpNaRaman1F1, AmpNaRaman1F2, ...
              AmpNaRaman1F1Ramp, AmpNaRaman1F2Ramp);
end

% Type < 0: no step is added
% 0 <= Type < 2: empty step with no light on
% 2 <= Type < 4: F2 only push out
% Type >= 4: F1 + F2 push out
Blast2Type = -2;
if (Blast2Type >= 0) && ~ImageOnly
    %% Blast (before any lowering)
    blurt('Blast without lowering');
    s.add('NaF2DP400/AMP', 1);
    TBlast2OP = 2e-3;
    DetBlast2OP = -9e6;
    if Blast2Type >= 2
        blurt('-- Including F2 blast');
        AmpNaF2DPBlast2OP = 0.2;
        AmpNaBlast2OP = 0.25;
```

```
    else
        AmpNaF2DPBlast2OP = 0;
        AmpNaBlast2OP = 0;
    end
    if Blast2Type >= 4
        blurt('-- Including F1+F2 blast');
        AmpNaF2DPBlast2OP = 0.3;
        AmpNaRepumpBlast2OP = 0.3;
    else
        AmpNaRepumpBlast2OP = 0;
    end
    s.addStep(@NaOpPrepStep, DetBlast2OP, AmpNaRepumpBlast2OP);
    s.addStep(@NaOPStep, TBlast2OP, AmpNaBlast2OP, AmpNaF2DPBlast2OP);
    s.add('NaF2DP400/AMP', 0);
end

TNaRaman2 = 0; 13e-6;
if (TNaRaman2 > 0) && ~ImageOnly
    %% Na Raman spectroscopy
    blurt('Raman spectroscopy after blasting, before lowering');
    NaRaman2Det = -12.255e6; -18.5e6; 605e6; -3.27e6;
    NaRaman2Delta = 0; -25e3; -20e3; -0.2e6;
    AmpNaRaman2F1 = 0.25; % 0.8 for coprop (max rabi rate); 0.4 for to match power with ran
    AmpNaRaman2F2 = 0.22;
    NaRaman2F1Vertical = 0;
    NaRaman2F2CounterOP = 0;
    s.addStep(@NaRamanPrepStep, NaRaman2F1Vertical, NaRaman2F2CounterOP);
    s.addStep(@NaRamanStep, TNaRaman2, NaRaman2Det, NaRaman2Delta, ...
        AmpNaRaman2F1, AmpNaRaman2F2);
end

TPumpD1 = 0;
if (TPumpD1 > 0) && ~ImageOnly
    s.addStep(TPumpD1) ...
        .add('NaD1/AMP', 0.12);
    s.addStep(1e-6) ...
        .add('NaD1/AMP', 0);
end

TDepumpOP = 0;
if (TDepumpOP > 0) && ~ImageOnly
    %% Depumping
    blurt('Depump');
    AmpF1DepumpOP = 0; 0.28;
    AmpF2D1DepumpOP = 0.21;
    if AmpF1DepumpOP > 0
        s.addStep(1e-6) ...
            .add('NaOP/AMP', 0.25);
    end
    s.addStep(TDepumpOP) ...
        .add('NaF1DP400/AMP', AmpF1DepumpOP) ...
        .add('NaD1/AMP', AmpF2D1DepumpOP);
    s.addStep(3e-6) ...
        .add('NaF1DP400/AMP', 0) ...
        .add('NaD1/AMP', 0) ...
        .add('NaOP/AMP', 0);
end
```

```
TDepump2OP = 0; 100e-6;
if (TDepump2OP > 0) && ~ImageOnly
    %% More depumping
    blurt('Additional depump');
    DetDepump2OP = -60e6;
    AmpNaF2DPDepump2OP = 0.5; % 30 uW on table
    AmpNaRepumpDepump2OP = 0; 0.25;
    AmpNaDepump2OP = 0.25;
    s.addStep(@NaOpPrepStep, DetDepump2OP, AmpNaRepumpDepump2OP);
    s.addStep(@NaOPStep, TDepump2OP, AmpNaDepump2OP, AmpNaF2DPDepump2OP);
end

TNaRaman3 = 0; 32e-6;
if (TNaRaman3 > 0) && ~ImageOnly
    %% Na Raman spectroscopy
    blurt('Raman spectroscopy');
    NaRaman3Det = -18.48e6; -18.476e6; -12.356e6;
    NaRaman3Delta = 0; -25e3; -20e3; -0.2e6;
    AmpNaRaman3F1 = 0.8;
    AmpNaRaman3F2 = 0.7;
    NaRaman3F1Vertical = 0; 1;
    NaRaman3F2CounterOP = 0; 1;
    s.addStep(@NaRamanPrepStep, NaRaman3F1Vertical, NaRaman3F2CounterOP);
    s.addStep(@NaRamanStep, TNaRaman3, NaRaman3Det, NaRaman3Delta, ...
        AmpNaRaman3F1, AmpNaRaman3F2);
end

TRNR = 0;
if TRNR > 0
    %% Release and recapture
    s.addStep(10e-6)...
        .add('TTLtweezerSnH',1);
    s.addStep(TRNR)...
        .add('TiSapph/AMP', 0);
    s.addStep(10e-6)...
        .add('TiSapph/AMP', 1);
    s.addStep(150e-6)...
        .add('TTLtweezerSnH', 0);
end

 TrapFreq = 0;
if TrapFreq > 0
    %% Trap frequency
    AmpModTweezer = .1;.05;.03;
    ModTweezer = TrapFreq * 2;
    if TrapFreq >= 150e3
        TModTweezer = 10000e6 / (ModTweezer^2);
    else
        TModTweezer = 10000e6 / (ModTweezer^2);TModTweezer = 100e6 / (ModTweezer^2) * 20;
    end
    if TModTweezer > 1, TModTweezer = 0; disp('TModTweezer too long!');beep;  end

    s.addStep(TModTweezer) ...
        .add('ModTisapphODTAOM/FREQ', 80e6 + ModTweezer)...
        .add('ModTisapphODTAOM/AMP', AmpModTweezer);
    s.wait(1e-5);
    s.add('ModTisapphODTAOM/AMP', 0);
end
```

```
s.add('TTLpiezomirror', 1);

% s.addStep(10e-3) ...
%     .add('TTLSwitchToggleTiSapph', 0);

% Adiabatic Lowering Parameters
% For either pushout or lowering measurement
% 0.03 is used for pushout.
% 0.036 gives 1.85 mW out of the fiber with AC tweezer, which is I assume
% what the pushout value should be from the calibrations.
TLowerRamp = 0.1e-3;
VTweezerNaLower = 0.05;
TRaiseRamp = 20e-3; TLowerRamp; % TLowerRamp; %Set to 0 to skip.
Will only happen if lowered first.
if (TLowerRamp > 0) && (~ImageOnly || AllowLowering)
    %% Adiabatic Lowering of tweezer
    blurt('Lower tweezer');
    % s.addStep(TLowerRamp) ...
    %     .add('VTweezerNa', LowerAdiabatically(VTweezerNaLower + VTweezerOffset));
    s.addStep(TLowerRamp) ...
        .add('VTweezerNa', linearRamp(VTweezerNaRaman, VTweezerNaLower + VTweezerOffset));
end

% Wait time for lowering in order to make sure axial excited states can
% leave the trap.
s.wait(5e-3);

%% push out
% Type < 0: no step is added
% 0 <= Type < 2: empty step with no light on
% 2 <= Type < 4: F2 only push out
% Type >= 4: F1 + F2 push out
BlastType = -2;
if (BlastType >= 0) && ~ImageOnly
    %% Blast after lowering
    blurt('Blast after lowering');
    s.add('NaF2DP400/AMP', 1);
    % OP blasting parameters
    TBlastOP = 2e-3;
    DetBlastOP = -9e6; 2e6; % -9 for With B field, 2 for no B field
    if BlastType >= 2
        blurt('-- Include F2 blast');
        AmpNaF2DPBlastOP = 0.2;
        AmpNaBlastOP = 0.25;
    else
        AmpNaF2DPBlastOP = 0;
        AmpNaBlastOP = 0;
    end
    if BlastType >= 4
        AmpNaF2DPBlastOP = 0.3;
        AmpNaRepumpBlastOP = 0.3;
        blurt('-- Include F1+F2 blast');
    else
        AmpNaRepumpBlastOP = 0;
    end
    s.addStep(@NaOpPrepStep, DetBlastOP, AmpNaRepumpBlastOP);
    s.addStep(@NaOPStep, TBlastOP, AmpNaBlastOP, AmpNaF2DPBlastOP);
```

```
    s.add('NaF2DP400/AMP', 0);
end

%% ramp tweezer back up
if (TRaiseRamp > 0) && (TLowerRamp > 0) && (~ImageOnly || AllowLowering)
    blurt('Ramp tweezer back up');
    s.wait(1e-6);
    % s.addStep(TRaiseRamp) ...
    %     .add('TiSapph/AMP', linearRamp(VTweezerNaLower, 1));
    s.addStep(TRaiseRamp) ...
        .add('VTweezerNa', LowerAdiabatically(VTweezerNaRaman));
end

if VTweezerNa ~= VTweezerNaRaman
    s.addStep(10e-3) ...
        .add('VTweezerNa', rampTo(VTweezerNa));
end

%% Temporary
% s.addStep(10e-3) ...
%     .add('TTLSwitchToggleTiSapph', 1);

%% Turn off the bias field along X direction
s.addBackground(@SetVBFieldSubSequence, 0, VShimZero, 300e-6);
% This waits for both the B field change and the third image delay.
s.waitBackground();

if ThirdImage
    %% Third image
    blurt('Image');
    s.addStep(10e-6) ...
        .add('TTLSwitchToggleTiSapph',ACTweezerDuringImage) ...
        .add('NaF2DP110/FREQ', s.NaF2DP110Center - DetImage / 2) ...
        .add('NaF2DP110/AMP', AmpNaResDPImage) ...
        .add('NaF1DP400/AMP', AmpNaRPDPImage) ...
        .add('NaF2DP400/AMP', 1) ...
        .add('NaMOT/AMP', 0) ...
        .add('FreqCsLO', CsLOFreq(CsDetImage));
    s.wait(TImgPreTrigger);
    s.addStep(TImgPostTrigger) ...
        .add('TTLAndor',1);
    s.addStep(TImage) ...
        .add('NaMOT/AMP', AmpNaImage) ...
        .add('AmpCsRepump', CsAmpRPImage) ...
        .add('AmpCsCool', CsAmpImage);
    s.addStep(1e-5) ...
        .add('NaMOT/AMP', 0) ...
        .add('AmpCsRepump', 0) ...
        .add('AmpCsCool', 0);
    s.wait(1e-5);
    s.addStep(TImgPostOff) ...
        .add('TTLAndor',0);
end

%% Wait for the sequence to finish and an extra 10ms so that turning the
% MOT back on doesn't interfer with whatever we were doing. (e.g. camera
% triggers)
s.waitAll();
```

```
s.wait(5e-3);

s.addStep(@NaRamanFinishStep);

%% Set laser back to MOT setting
s.add('NaF1DP400/AMP', 1);
s.add('NaF2DP400/AMP', 1);
s.add('NaMOT/AMP', AmpMOT);
s.add('NaF2DP110/AMP', AmpNaResDPMOT);
s.add('NaF2DP110/FREQ', s.NaF2DP110Center - DetMOT/2);
s.add('TTLpiezomirror', 1);
s.addStep(@SetVBFieldSubSequence, VMOT, [Vx, Vy, Vz], 1e-3);
s.add('VTweezerNa', VTweezerNa);
s.add('TTLSwitchToggleTiSapph',1);
s.add('TTLSwitchToggleNa', 1);
s.add('NaD1/AMP', 0);

s.add('FreqCsLO', CsLOFreq(CsDetMOT));
s.add('AmpCsRepump', CsAmpRPMOT);
s.add('AmpCsCool', CsAmpMOT);
%s.add('AmpCsCool', 0);

s.run();
```

**end**

---


[1] J. Dalibard, Y. Castin, and K. Mølmer, Phys. Rev. Lett. **68**, 580 (1992), URL https://link.aps.org/doi/10.1103/PhysRevLett.68.580.

[2] R. Chrétien, *Laser cooling of atoms: Monte-Carlo wavefunction simulations* (2014), URL http://www.oq.ulg.ac.be/master_thesis_rc.pdf.

[3] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, SIAM Review **59**, 65 (2017), ISSN 0036-1445 (print), 1095-7200 (electronic).

[4] note about modification to the quantum jump method (https://goo.gl/EZ13wt)

[5] Library code (https://goo.gl/WLJwJp)

[6] Top level code (https://goo.gl/XXVreq)